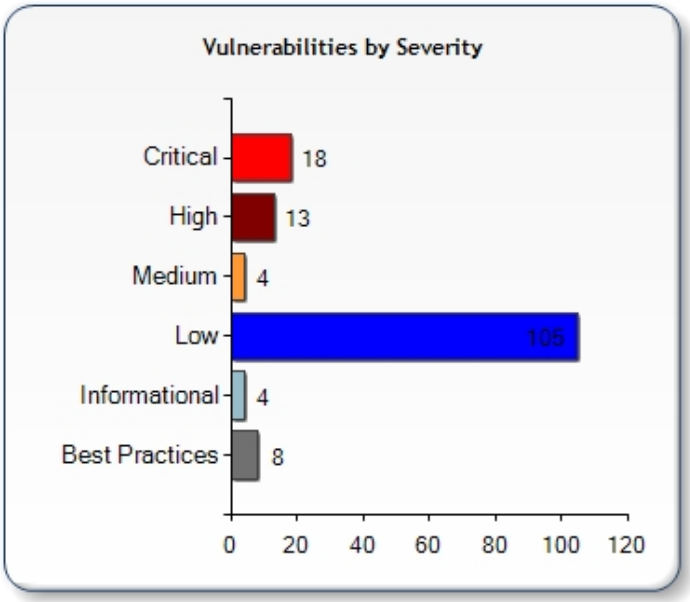
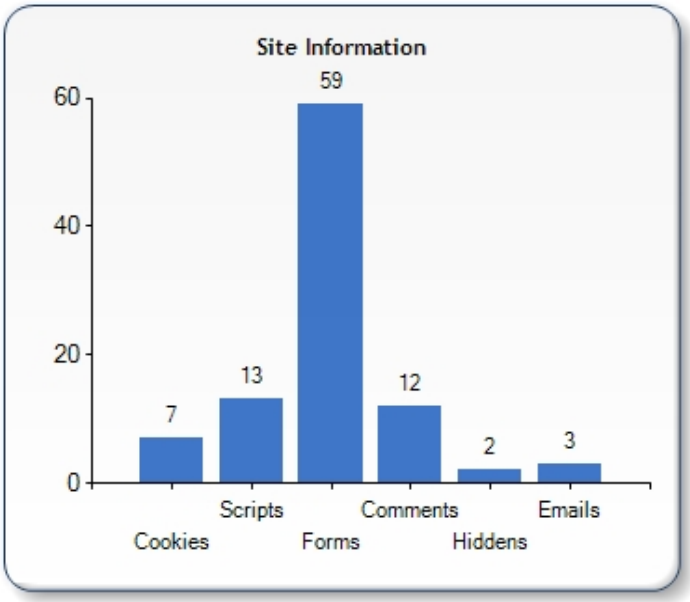
Web Application Assessment Report

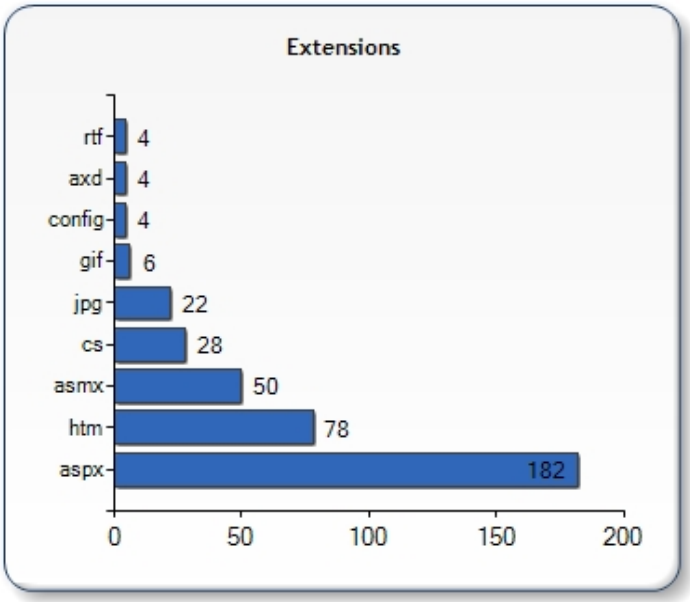
HP WebInspect™

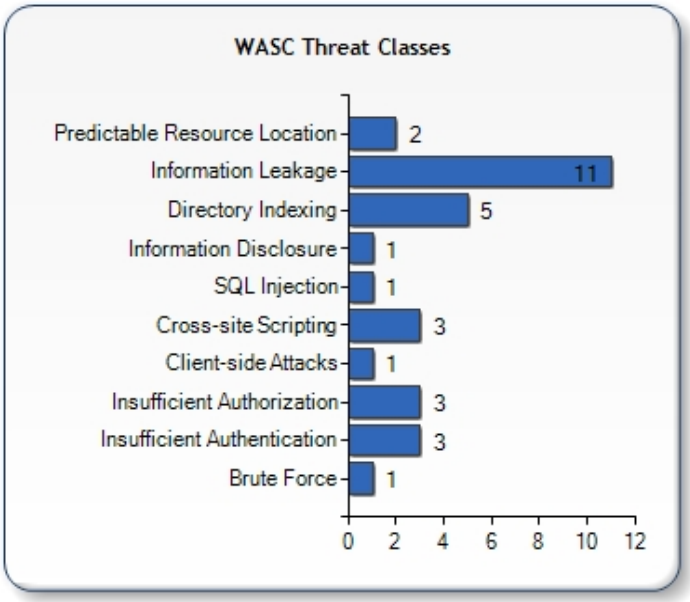
Scan: http://demo.testfire.net/
Policy: OWASP Top 10

Scan Date: Wednesday, February 04, 2009 2:37:05 PM
Scan Version: 7.7.869.0





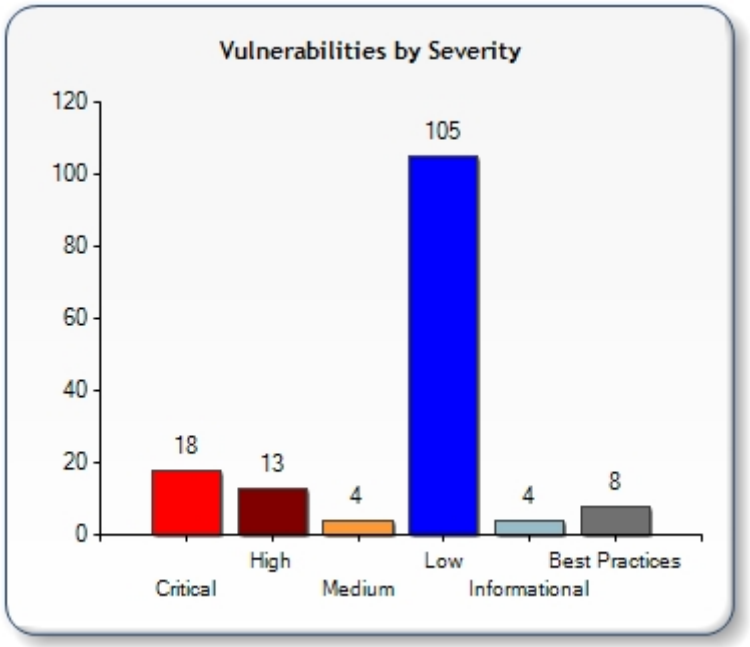




Scan: http://demo.testfire.net/
Policy: OWASP Top 10

Scan Date: Wednesday, February 04, 2009 2:37:05 PM
Scan Version: 7.7.869.0

Server: demo.testfire.net



Critical **Database Server Error Message**

File Names:
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx

Summary: Critical database server error message vulnerabilities were identified in the web application, indicating that an unhandled exception was generated in your web application code. Unhandled exceptions are circumstances in which the application has received user input that it did not expect and does not know how to handle. When successfully exploited, an attacker can gain unauthorized access to the database by using the information recovered from seemingly innocuous error messages to pinpoint flaws in the web application and to discover additional avenues of attack. Recommendations include designing and adding consistent error-handling mechanisms that are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Description

The most common cause of an unhandled exception is a failure to properly sanitize client-supplied data that is used in SQL statements. They can also be caused by a bug in the web application's database communication code, a misconfiguration of database connection settings, an unavailable database, or any other reason that would cause the application's database driver to be unable to establish a working session with the server. The problem is not that web applications generate errors. All web applications in their normal course of operation will at some point receive an unhandled exception. The problem lies not in that these errors were received, but rather in how they are handled. Any error handling solution needs to be well-designed, and uniform in how it handles errors. For instance, assume an attacker is attempting to access a specific file. If the request returns an error File not Found, the attacker can be relatively sure the file does not exist. However, if the error returns "Permission Denied," the attacker has a fairly good idea that the specific file does exist. This can be helpful to an attacker in many ways, from determining the operating system to discovering the underlying architecture and design of the application.

The error message may also contain the location of the file that contains the offending function. This may disclose the webroot's absolute path as well as give the attacker the location of application "include" files or database configuration information. A fundamental necessity for a successful attack upon your web application is reconnaissance. Database server error messages can provide information that can then be utilized when the attacker is formulating his next method of attack. It may even disclose the portion of code that failed.

Be aware that this check is part of unknown application testing which seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or remediation information for this issue. Please note that this vulnerability may be a false positive if the page it is flagged on is technical documentation relating to a database server.

Execution: The ways in which an attacker can exploit the conditions that caused the error depend on its cause. In the case of SQL injection, the techniques that are used will vary from database server to database server, and

even query to query. An in-depth guide to SQL Injection attacks is available at http://products.spidynamics.com/asclabs/sql_injection.pdf, or in the SQL Injection vulnerability information, accessible via the Policy Manager. Primarily, the information gleaned from database server error messages is what will allow an attacker to conduct a successful attack after he combines his various findings.

Implication:

The severity of this vulnerability depends on the reason that the error message was generated. In most cases, it will be the result of the web application attempting to use an invalid client-supplied argument in a SQL statement, which means that SQL injection will be possible. If so, an attacker will at least be able to read the contents of the entire database arbitrarily. Depending on the database server and the SQL statement, deleting, updating and adding records and executing arbitrary commands may also be possible. If a software bug or bug is responsible for triggering the error, the potential impact will vary, depending on the circumstances. The location of the application that caused the error can be useful in facilitating other kinds of attacks. If the file is a hidden or include file, the attacker may be able to gain more information about the mechanics of the web application, possibly even the source code. Application source code is likely to contain usernames, passwords, database connection strings and aids the attacker greatly in discovering new vulnerabilities.

Fix:

For Development:

From a development perspective, the best method of preventing problems from arising from database error messages is to adopt secure programming techniques that prevent problems that might arise from an attacker discovering too much information about the architecture and design of your web application. The following recommendations can be used as a basis for that.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad. Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

For Security Operations:

The following recommendations will help in implementing a secure database protocol for your web application. Be advised each database has its own method of secure lock down.

- **ODBC Error Messaging:** Turn off ODBC error messaging in your database server. Never display raw ODBC or other errors to the end user. See Removing Detailed Error Messages below, or consult your database server's documentation, for more information.
- **Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- **Proper Error Handling:** Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.
- **Stored Procedures:** Consider using stored procedures. They require a very specific parameter format, which makes them less susceptible to SQL Injection attacks.
- **Database Privileges:** Utilize a least-privileges scheme for the database application. Ensure that user accounts only have the limited functionality that is actually required. All database mechanisms should deny access until it has been granted, not grant access until it has been denied.

Removing Detailed Error Messages

Find instructions for turning off detailed error messaging in IIS at this link:

http://support.microsoft.com/kb/294807

Find information on suppressing error messages on an Apache server at the following locations:

Apache HTTP Server Version 1.3 Custom Error Responses

Apache HTTP Server Version 2.0 Custom Error Responses

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site. If the unhandled exception occurs in a piece of in-house developed software, consult the developer. If it is in a commercial package, contact technical support.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker?

Reference:

HP:

HP Application Security Center SQL Injection Whitepaper

Apache:

Apache HTTP Server Version 1.3 Custom Error Responses

Apache HTTP Server Version 2.0 Custom Error Responses

Microsoft:

Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes

Attack Request:

```
POST /bank/login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k5
5pfimdwanvfujx2midkm45;amSessionId=037914644

uid=%00&passw=foo&btnSubmit=Login
```

Attack Response:

```
HTTP/1.1 500 Internal Server Error
Connection: close
Date: Wed, 04 Feb 2009 06:43:42 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 5011



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_head"><title>
 Altoro Mutual: Server Error
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label for="txtSearch">Search</label>
         <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
         <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
    </tr>
   </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

< ... {content removed}
```

Critical **Cross-Site Scripting**

File Names:

- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/comment.aspx
- http://demo.testfire.net:80/search.aspx?txtSearch=12345%3csCrIpT%3ealert(48745)%3c%2fsCrIpT%3e
- http://demo.testfire.net:80/bank/login.aspx

Summary: Cross-Site Scripting vulnerabilities were verified as executing code on the web application. Cross-Site Scripting occurs when dynamically generated web pages display user input, such as login information, that is not properly validated, allowing an attacker to embed malicious scripts into the generated page and then

execute the script on the machine of any user that views the site. In this instance, the web application was vulnerable to an automatic payload, meaning the user simply has to visit a page to make the malicious scripts execute. If successful, Cross-Site Scripting vulnerabilities can be exploited to manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on end user systems. Recommendations include implementing secure programming techniques that ensure proper filtration of user-supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Execution:

View the attack string included with the request to check what to search for in the response. For instance, if "(javascript:alert('XSS')" is submitted as an attack (or another scripting language), it will also appear as part of the response. This indicates that the web application is taking values from the HTTP request parameters and using them in the HTTP response without first removing potentially malicious data.

Implication:

XSS can generally be subdivided into two categories: stored and reflected attacks. The main difference between the two is in how the payload arrives at the server. Stored attacks are just that...in some form stored on the target server, such as in a database, or via a submission to a bulletin board or visitor log. The victim will retrieve and execute the attack code in his browser when a request is made for the stored information. Reflected attacks, on the other hand, come from somewhere else. This happens when user input from a web client is immediately included via server-side scripts in a dynamically generated web page. Via some social engineering, an attacker can trick a victim, such as through a malicious link or "rigged" form, to submit information which will be altered to include attack code and then sent to the legitimate server. The injected code is then reflected back to the user's browser which executes it because it came from a trusted server. The implication of each kind of attack is the same.

The main problems associated with successful Cross-Site Scripting attacks are:

- Account hijacking - An attacker can hijack the user's session before the session cookie expires and take actions with the privileges of the user who accessed the URL, such as issuing database queries and viewing the results.
- Malicious script execution - Users can unknowingly execute JavaScript, VBScript, ActiveX, HTML, or even Flash content that has been inserted into a dynamically generated page by an attacker.
- Worm propagation - With Ajax applications, XSS can propagate somewhat like a virus. The XSS payload can autonomously inject itself into pages, and easily re-inject the same host with more XSS, all of which can be done with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user.
- Information theft - Via redirection and fake sites, attackers can connect users to a malicious server of the attacker's choice and capture any information entered by the user.
- Denial of Service - Often by utilizing malformed display requests on sites that contain a Cross-Site Scripting vulnerability, attackers can cause a denial of service condition to occur by causing the host site to query itself repeatedly .
- Browser Redirection - On certain types of sites that use frames, a user can be made to think that he is in fact on the original site when he has been redirected to a malicious one, since the URL in the browser's address bar will remains the same. This is because the entire page isn't being redirected, just the frame in which the JavaScript is being executed.
- Manipulation of user settings - Attackers can change user settings for nefarious purposes.

Fix:

For Development:

Cross-Site Scripting attacks can be avoided by carefully validating all input, and properly encoding all output. Validation can be done using standard ASP.NET Validation controls, or directly in your code. Always use as strict a pattern as you can possibly allow.

Encoding of output ensures that any scriptable content is properly encoded for HTML before being sent to the client. This is done with the function HttpUtility.HtmlEncode, as shown in the following Label control sample:

```
Label2.Text = HttpUtility.HtmlEncode(input)
```

Be sure to consider all paths that user input takes through your application. For instance, if data is entered by the user, stored in a database, and then redisplayed later, you must make sure it is properly encoded each time it is retrieved. If you must allow free-format text input, such as in a message board, and you wish to allow some HTML formatting to be used, you can handle this safely by explicitly allowing only a small list of safe tags. Here are examples of how to do this safely:

C# Example:

```
StringBuilder sb = new StringBuilder(
    HttpUtility.HtmlEncode(input));
sb.Replace("&lt;b&gt;", "<b>");
sb.Replace("&lt;/b&gt;", "<b>");
sb.Replace("&lt;i&gt;", "<i>");
sb.Replace("&lt;/i&gt;", "</i>");
Response.Write(sb.ToString());
```

VB.NET Example:

```
Dim sb As StringBuilder = New StringBuilder( _
    HttpUtility.HtmlEncode(input))
sb.Replace("&lt;b&gt;", "<b>")
sb.Replace("&lt;/b&gt;", "<b>")
sb.Replace("&lt;i&gt;", "<i>")
sb.Replace("&lt;/i&gt;", "</i>")
Response.Write(sb.ToString())
```

Java Example:

```
public static String HTMLEncode(String aTagFragment){
final StringBuffer result = new StringBuffer();
```

```
final StringCharacterIterator iterator = new StringCharacterIterator(aTagFragment);
char character = iterator.current();
while (character != StringCharacterIterator.DONE ){
if (character = = '<') {
result.append("&lt;");
}
else if (character = = '>') {
result.append("&gt;");
}
else if (character = = '\"') {
result.append("&quot;");
}
else if (character = = '\') {
result.append("&#039;");
}
else if (character = = '\\') {
result.append("&#092;");
}
else if (character = = '&') {
result.append("&amp;");
}
else {
//the char is not a special one
//add it to the result as is
result.append(character);
}
character = iterator.next();
}
return result.toString();
}
```

The following recommendations will help you build web applications capable of withstanding Cross-Site Scripting attacks.

- Define what is allowed. Ensure that the web application validates all input parameters (cookies, headers, query strings, forms, hidden fields, etc.) against a stringent definition of expected results.
- Check the responses from POST and GET requests to ensure what is being returned is what is expected, and is valid.
- Remove conflicting characters, brackets, and single and double quotes from user input by encoding user supplied data. This will prevent inserted scripts from being sent to end users in a form that can be executed.
- Whenever possible, limit all client-supplied data to alphanumeric data. Using this filtering scheme, if a user entered " <script>alertdocumentcookie('aaa') </script>", it would be reduced to "scriptalertdocumentcookiescript". If non-alphanumeric characters must be used, encode them as HTML entities before using them in an HTTP response, so that they cannot be used to modify the structure of the HTML document.
- Use two-factor customer authentication mechanisms as opposed to single-factor authentication.
- Verify the origin of scripts before you modify or utilize them.
- Do not implicitly trust any script given to you by others (whether downloaded from the web, or given to you by an acquaintance) for use in your own code.

Most server side scripting languages provide built in methods to convert the value of the input variable into correct, non-interpretable HTML. These should be used to sanitize all input before displayed to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP / ASP.NET: Server.HTMLEncode (strHTML String)

For Security Operations:

Server-side encoding, where all dynamic content is first sent through an encoding function where Scripting tags will be replaced with codes in the selected character set, can help to prevent Cross-Site Scripting attacks. The drawback to server-side encoding is that it can be resource intensive, and may have a negative performance impact on some web servers.

If site users must be allowed to use HTML tags, such as a bulletin board where the user would be allowed to use formatting tags, limit the ones that can be used. Create a list of acceptable tags, such as bold, italic or underline, and only allow those to be used. Any other tags should be rejected. Below are a few regular expressions that will help detect Cross-Site Scripting.

Regex for a simple CSS attack:

```
/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/ix
```

The above regular expression would be added into a new Snort rule as follows:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII Cross-Site Scripting attempt";
flow:to_server,established; pcre:"/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/i";
classtype:Web-application-attack; sid:9000; rev:5;)
```

Paranoid regex for CSS attacks:

```
/((\%3C)|<)[^\n]+((\%3E)|>)/I
```

This signature simply looks for the opening HTML tag, and its hex equivalent, followed by one or more characters other than the new line, and then followed by the closing tag or its hex equivalent. This may end

up giving a few false positives depending upon how your Web application and Web server are structured, but it is guaranteed to catch anything that even remotely resembles a Cross-Site Scripting attack. From a public perspective, you can also strengthen educational programs to help consumers avoid online scams, such as phishing, that can be utilized in account hijackings and other forms of identity theft.

For QA:

Fixes for Cross-Site Scripting defects will ultimately require code based fixes. The steps detailed in the Developer and Security Operations section will provide any developer with the information necessary to remediate these issues. The following steps outline how to manually test an application for Cross-Site Scripting.

Step 1. Open any Web site in a browser, and look for places on the site that accept user input such as a search form or some kind of login page. Enter the word test in the search box and send this to the Web server.

Step 2. Look for the Web server to respond back with a page similar to something like "Your search for 'test' did not find any items" or "Invalid login test." If the word "test" appears in the results page, you are in luck.

Step 3. To test for Cross-Site Scripting, input the string "<script>alert('hello')</script>" without the quotes in the same search or login box you used before and send this to your Web server.

Step 4. If the server responds back with a popup box that says "hello", then the site is vulnerable to Cross-Site Scripting.

Step 5. If Step 4 fails and the Web site does not return this information, you still might be at risk. Click the 'View Source' option in your browser so you can see the actual HTML code of the Web page. Now find the <script> string that you sent the server. If you see the entire "<script>alert('hello')</script>" text in this source code, then the Web server is vulnerable to Cross-Site Scripting.

Reference:

SPI Dynamics Cross-Site Scripting Whitepaper
http://products.spidynamics.com/asclabs/cross-site_scripting.pdf

OWASP Cross-Site Scripting Information
http://www.owasp.org/documentation/topten/a4.html

Microsoft
http://support.microsoft.com/default.aspx?scid=kb;EN-US;q252985

Microsoft Anti-Cross Site Scripting Library V1.0
http://www.microsoft.com/downloads/details.aspx?familyid=9a2b9c92-7ad9-496c-9a89-af08de2e5982&displaylang=en

CERT
http://www.cert.org/advisories/CA-2000-02.html

Apache
http://httpd.apache.org/info/css-security/apache_specific.html

Netscape
http://channels.netscape.com/ns/browsers/security.jsp

SecurityFocus.com
http://www.securityfocus.com/infocus/1768

Attack Request:

```
POST /subscribe.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/subscribe.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 73
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k55pfimdwanvfujx2midkm45;amSessionId=037914644

txtEmail=12345%3csCrIpT%3ealert(51551)%3c%2fsCrIpT%3e&btnSubmit=Subscribe
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:46:30 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8731



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Event Subscription
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
```

rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Mutual, subscription, mailing list"></head>
<body style="margin-top:5px;">

<form id="frmSearch" method="get" action="/search.aspx">
<table width="100%" border="0" cellpadding="0" cellspacing="0">
<tr>
<td rowspan="2"></td>
<td align="right" valign="top">
Sign In | Contact Us | Feedback | <label for="txtSearch">Search</label>
<input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
<input type="submit" value="Go" />
</td>
</tr>
<tr>
<td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"></td>
</tr>
</table>
</form>

<table cell ... {content removed}

Critical **SQL Injection Confirmed (No Data Extraction)**

File Names:
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/subscribe.aspx

Summary: Critical SQL Injection vulnerabilities have been identified in the web application. SQL Injection is a method of attack where an attacker can exploit vulnerable code and the type of data an application will accept, and can be exploited in any application parameter that influences a database query. Examples include parameters within the url itself, post data, or cookie values. If successful, SQL Injection can give an attacker access to backend database contents, the ability to remotely execute system commands, or in some circumstances the means to take control of the server hosting the database. Recommendations include employing a layered approach to security that includes utilizing parameterized queries when accepting user input, ensuring that only expected data is accepted by an application, and hardening the database server to prevent data from being accessed inappropriately.

Be advised that database extraction could not performed. This could be due to one of several reason:

- The version of the database is not supported, such as versions of MySQL prior to 5.0. These versions do not have standard system tables to store database metadata information such as user table names or column names that are required to extract user data.
- The database does not allow extraction (MSACCESS) which by default does not allow access to system metadata tables to user.
- The injection could be in a clause that does not allow automated extraction (cannot predict union or in order by clause, etc.).

An attacker can still perform brute force attacks to extract database metadata information along with conducting more serious attacks such as executing a DROP TABLE command or modifying data inside the database.

Execution: Consider a login form for a web application. If the user input from the form is directly utilized to build a dynamic SQL statement, then there has been no input validation conducted, giving control to an attacker who wants access to the database. Basically, an attacker can use an input box to send their own request to the server, and then utilize the results in a malicious manner. This is a very typical scenario considering that HTML pages often use the POST command to send parameters to another ASP page. The number in bold might be supplied by the client in an HTTP GET or POST parameter, like in the following URL:

http://www.example.com/GetItemPrice?ItemNumber=**12345**

In the example above, the client-supplied value, **12345**, is simply used as a numeric expression to indicate the item that the user wants to obtain the price of an item. The web application takes this value and inserts it into the SQL statement in between the single quotes in the WHERE clause. However, consider the following URL:

http://www.example.com/GetItemPrice?ItemPrice?ItemNumber=0' UNION SELECT CreditCardNumber FROM Customers WHERE '1'='1

In this case, the client-supplied value has actually modified the SQL statement itself and 'injected' a statement of his or her choosing. Instead of the price of an item, this statement will retrieve a customer's credit card number.

Implication: Fundamentally, SQL Injection is an attack upon the web application, not the web server or the operating system itself. As the name implies, SQL Injection is the act of adding an unexpected SQL commands to a query, thereby manipulating the database in ways unintended by the database administrator or developer. When successful, data can be extracted, modified, inserted or deleted from database servers that are used

by vulnerable web applications. In certain circumstances, SQL Injection can be utilized to take complete control of a system.

Fix:

Each method of preventing SQL injection has its own limitations. Therefore, it is wise to employ a layered approach to preventing SQL injection, and implement several measures to prevent unauthorized access to your backend database. The following are recommended courses of action to take to prevent SQL Injection and Blind SQL Injection vulnerabilities from being exploited in your web application.

For Development:

Use the following recommendations to code web applications that are not susceptible to SQL Injection attacks.

- **Parameterized Queries:** SQL Injection arises from an attacker's manipulation of query data to modify query logic. The best method of preventing SQL Injection attacks is thereby to separate the logic of a query from its data. This will prevent commands inserted from user input from being executed. The downside of this approach is that it can have an impact on performance, albeit slight, and that each query on the site must be structured in this method for it to be completely effective. If one query is inadvertently bypassed, that could be enough to leave the application vulnerable to SQL Injection. The following code shows a sample SQL statement that is SQL injectable.

  ```
  sSql = "SELECT LocationName FROM Locations ";
  sSql = sSql + " WHERE LocationID = " + Request["LocationID"];
  oCmd.CommandText = sSql;
  ```

  The following example utilizes parameterized queries, and is safe from SQL Injection attacks.

  ```
  sSql = "SELECT * FROM Locations ";
  sSql = sSql + " WHERE LocationID = @LocationID";
  oCmd.CommandText = sSql;
  oCmd.Parameters.Add("@LocationID", Request["LocationID"]);
  ```

  The application will send the SQL statement to the server without including the user's input. Instead, a parameter-@LocationID- is used as a placeholder for that input. In this way, user input never becomes part of the command that SQL executes. Any input that an attacker inserts will be effectively negated. An error would still be generated, but it would be a simple data-type conversion error, and not something which a hacker could exploit.

  The following code samples show a product ID being obtained from an HTTP query string, and used in a SQL query. Note how the string containing the "SELECT" statement passed to SqlCommand is simply a static string, and is not concatenated from input. Also note how the input parameter is passed using a SqlParameter object, whose name ("@pid") matches the name used within the SQL query.

  C# sample:

  ```
  string connString = WebConfigurationManager.ConnectionStrings["myConn"].ConnectionStrin
  g;
  using (SqlConnection conn = new SqlConnection(connString))
  {
     conn.Open();
     SqlCommand cmd = new SqlCommand("SELECT Count(*) FROM Products WHERE ProdID=
  @pid", conn);
     SqlParameter prm = new SqlParameter("@pid", SqlDbType.VarChar, 50);
     prm.Value = Request.QueryString["pid"];
     cmd.Parameters.Add(prm);
     int recCount = (int)cmd.ExecuteScalar();
  }
  ```

  VB.NET sample:

  ```
  Dim connString As String = WebConfigurationManager.ConnectionStrings("myConn").Connecti
  onString
  Using conn As New SqlConnection(connString)
     conn.Open()
     Dim cmd As SqlCommand = New SqlCommand("SELECT Count(*) FROM Products WHERE
  ProdID=@pid", conn)
     Dim prm As SqlParameter = New SqlParameter("@pid", SqlDbType.VarChar, 50)
     prm.Value = Request.QueryString("pid")
     cmd.Parameters.Add(prm)
     Dim recCount As Integer = cmd.ExecuteScalar()
  End Using
  ```

- **Validate input:** The vast majority of SQL Injection checks can be prevented by properly validating user input for both type and format. The best method of doing this is via "white listing". This is defined as only accepting specific account numbers or specific account types for those relevant fields, or only accepting integers or letters of the English alphabet for others. Many developers will try to validate input by "black listing" characters, or "escaping" them. Basically, this entails rejecting known bad data, such as a single quotation mark, by placing an "escape" character in front of it so that the item that follows will be treated as a literal value. This approach is not as effective as white listing because it is impossible to know all forms of bad data ahead of time.

For Security Operations:

Use the following recommendations to help prevent SQL Injection attacks upon your web applications.

- **Restrict Application Privileges:** Limit user credentials so that only those rights the application needs to function are utilized. Any successful SQL Injection attack would run in the context of the

user's credential. While limiting privileges will not prevent SQL Injection attacks outright, it will make them significantly harder to enact.

- **Strong SA Password Policy:** Often, an attacker will need the functionality of the administrator account to utilize specific SQL commands. It is much easier to "brute force" the SA password when it is weak, and will increase the likelihood of a successful SQL Injection attack. Another option is not to use the SA account at all, and instead create specific accounts for specific purposes.
- **Consistent Error Messaging Scheme:** Ensure that you provide as little information to the user as possible when a database error occurs. Don't reveal the entire error message. Error messages need to be dealt with on both the web and application server. When a web server encounters a processing error it should respond with a generic web page, or redirect the user to a standard location. Debug information, or other details that could be useful to a potential attacker, should never be revealed. Application servers, like WebSphere, often install with error messages or debug settings enabled by default. Consult your application server's documentation for information on suppressing those error messages.
- **Stored Procedures:** If unused, delete SQL stored procedures such as master..Xp_cmdshell, xp_startmail, xp_sendmail, and sp_makewebtask.

SQL Injection vulnerabilities are inherently tied to the actual code of your web application. While not a fix, you can implement an emergency measure by adding a rule that incorporates a regular expression to your IDS to check for SQL Injection attacks. While this will not resolve all possible SQL injection vulnerabilities, it is simple to implement, and will require an attacker to escalate his methodology to achieve a successful attack. Regular expressions that can be utilized to do this follow.

Regex for detection of SQL meta-characters:
/(\%27)|(\')|(\-\-)|(\%23)|(#)/ix

The above regular expression would be added into a Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection - Paranoid"; flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(\')|(\-\-)|(%23)|
(#)/i"; classtype:Web-application-attack; sid:9099; rev:5;)

Regex for typical SQL Injection attacks:
/\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

Regex for detecting SQL Injection with the UNION keyword:
/((\%27)|(\'))union/ix
(\%27)|(\')
the single-quote and its hex equivalent union - the keyword union

Similar expressions can be written for other SQL queries such as select, insert, update, delete, drop, and so on.

Regex for detecting SQL Injection attacks on a MS SQL Server:
/exec(\s|\+)+(s|x)p\w+/ix

For QA:
Fixes for SQL Injection defects will ultimately require code based fixes. The steps detailed in the Developer and Security Operations section will provide any developer with the information necessary to remediate these issues. The following steps outline how to manually test an application for SQL Injection.

How to manually test applications for SQL Injection:

1. Open the web application you wish to test for SQL Injection defects in a browser.

2. Mouse over the links of the Web site with your cursor while paying attention to the bottom status bar. You will notice the URLs that the links point to. Try to find a URL with parameters in it. Ex. http://www.site.com/articleid.asp?id=42.

Note: If you don't see any URL's in the status bar, then just click on links and watch the address bar until you find a URL that has parameters.

3. Once a URL with parameters has been found, click the link and go to that page. In the Address bar you should now see the URL that was seen in the status bar.

4. There are two methods for testing scripts for SQL injection. Be sure to test each parameter value one at a time with both methods.

Method 1. Go to the address bar, click your cursor, and highlight a parameter value Ex. Highlight the word value in "name=value" and replace it with a single quote (').It should now look like "name=' "

Method 2. Go to the address bar, click your cursor, and put a single quote (') in the middle of the value. It should now look like "name=val'ue"

5. Click the 'GO' button. This will send your request to the Web server.

6. Analyze the response from the Web server for any error messages. Most database error messages will look similar to the examples below:

Example error 1:
Microsoft OLE DB Provider for SQL Server error '80040e14'
Unclosed quotation mark before the character string '51 ORDER BY
some_name'. /some_directory/some_file.asp, line 5

Example error 2:
ODBC Error Code = S1000 (General error)
[Oracle][ODBC][Ora]ORA-00933: SQL command not properly ended

Example error 3:
Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)
Message: View's SELECT and view's field list have different column counts

7. Sometimes the error message is not obvious and is hidden in the source of the page. To look for it, you must view the HTML source of the page and search for the error. To do this in Internet Explorer, click the 'View' menu, and select the 'Source' option. This will cause notepad to open with the HTML source of the page. In notepad, click the 'Edit' menu and select 'Find'. A dialog box will appear that will ask you to 'Find What'. Type the phrase 'Microsoft OLE DB' or '[ODBC]' and click 'Find Next'.

8. If either step 6 or 7 is successful, then the Web site is vulnerable to SQL injection.

Reference:

SPI Dynamics SQL Injection Whitepaper
http://products.spidynamics.com/asclabs/sql_injection.pdf

SPI Dynamics Blind SQL Injection Whitepaper
http://products.spidynamics.com/asclabs/blind_sql_injection.pdf

Microsoft
http://msdn.microsoft.com/msdnmag/issues/04/09/SQLInjection/default.aspx
http://support.microsoft.com/default.aspx?scid=kb;en-us;302570

SQLSecurity.com
http://www.sqlsecurity.com/DesktopDefault.aspx

OWASP
http://www.owasp.org/index.php/SQL_Injection

Attack Request:

```
POST /bank/login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 90
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641

uid=12345'+and++(select+count(*)+from+spitable)%3d1+or+'1'%3d'0+&passw=foo&btnSubmit=Login
```

Attack Response:

```
HTTP/1.1 500 Internal Server Error
Connection: close
Date: Wed, 04 Feb 2009 06:44:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 5181



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_head"><title>
 Altoro Mutual: Server Error
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label for="txtSearch">Search</label>
         <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
         <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
    </tr>
   </table>
 </form>
```

```
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

< ... {content removed}
```

High **Logins Sent Over Unencrypted Connection**

File Names:
- http://demo.testfire.net:80/bank/Login.aspx
- http://demo.testfire.net:80/admin/Login.aspx
- http://demo.testfire.net:80/bank/login.aspx

Summary: Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. http://demo.testfire.net:80/bank/Login.aspx has failed this policy. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication: An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execition of actions not intended by the application developers.

Fix:

For Security Operations:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For Development:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For QA:

Test the application not only from the perspective of a normal user, but also from the perspective of a malicious one.

Attack Request:

```
GET /bank/Login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:47:56 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8729



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Mutual
Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
```

```
id="_ctl0__ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
        <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
        <input type="submit" value="Go" />
    </td>
   </tr>
   <tr>
    <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
   </tr>
  </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">

 ... {content removed}
```

High **Microsoft ASP.NET Request Filtering Bypass Cross-Site Scripting Vulnerability**

File Names:
- http://demo.testfire.net:80/search.aspx?txtSearch="></XSS/*-*/STYLE=xss:e/**/xpression(alert(097531))>
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/comment.aspx

Summary: A Cross-Site Scripting vulnerability has been detected in Microsoft ASP.NET request filtering. Web applications that are coded in any .NET language and rely only on the default .NET request filtering are vulnerable to Cross-Site Scripting. If exploited, an attacker can manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on end user systems. Recommendations include implementing secure programming techniques that ensure proper filtration of user-supplied data, and encoding all user-supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Execution: The following is a proof of concept:

http://www.example.com/MyApp.aspx?myvar=</XSS/*-*/STYLE=xss:e/**/xpression(alert('XSS'))>

Implication: The main problems associated with successful Cross-Site Scripting attacks are:

- Account hijacking - An attacker can hijack the user's session before the session cookie expires and take actions with the privileges of the user who accessed the URL, such as issuing database queries and viewing the results.
- Malicious script execution - Users can unknowingly execute JavaScript, VBScript, ActiveX, HTML, or even Flash content that has been inserted into a dynamically generated page by an attacker.
- Worm propagation - With Ajax applications, XSS can propagate somewhat like a virus. The XSS payload can autonomously inject itself into pages, and easily re-inject the same host with more XSS, all of which can be done with no hard refresh. Thus, XSS can send multiple requests using complex HTTP methods to propagate itself invisibly to the user.
- Information theft - Via redirection and fake sites, attackers can connect users to a malicious server of the attacker's choice and capture any information entered by the user.
- Denial of Service - Often by utilizing malformed display requests on sites that contain a Cross-Site Scripting vulnerability, attackers can cause a denial of service condition to occur by causing the host site to query itself repeatedly .
- Browser Redirection - On certain types of sites that use frames, a user can be made to think that he is in fact on the original site when he has been redirected to a malicious one, since the URL in the browser's address bar will remains the same. This is because the entire page isn't being redirected, just the frame in which the JavaScript is being executed.
- Manipulation of user settings - Attackers can change user settings for nefarious purposes.

For more detailed information on Cross-Site Scripting attacks, see the SPI Dynamics Cross-Site Scripting whitepaper.

Vulnerable products and versions:

- Microsoft Windows Server 2003 Standard Edition Build 3790.srv03_sp1_rtm.050324-1447 Service Pack 1
- Microsoft IIS 6.0
- Microsoft ASP .NET Framework Version 2.0.50727.42
- Microsoft Internet Explorer 6.0.2900.2180.xpsp_sp2_gdr.050301-1519
- Microsoft Internet Explorer 7.0.5450.4 Beta 3
- Microsoft Internet Explorer 7.0.5730.11

Note: We were unable to confirm the vulnerability in the latest versions of Microsoft ASP .NET Framework and Microsoft Internet Explorer.

Fix: **For Development:**

Cross-Site Scripting attacks can be avoided by carefully validating all input, and properly encoding all output. Do not rely solely on default ASP.NET validation controls. In addition to using the default validation controls, properly sanitize all input parameters on server side applications by adopting a whitelist strategy to input validation. You may also validate input parameters directly in your code. Always use as strict a pattern as you can possibly allow.

Encoding of output ensures that any scriptable content is properly encoded for HTML before being sent to the client. This is done with the function HttpUtility.HtmlEncode, as shown in the following Label control sample:

Label2.Text = HttpUtility.HtmlEncode(input)

Be sure to consider all paths that user input takes through your application. For instance, if data is entered by the user, stored in a database, and then redisplayed later, you must make sure it is properly encoded each time it is retrieved. If you must allow free-format text input, such as in a message board, and you wish to allow some HTML formatting to be used, you can handle this safely by explicitly allowing only a small list of safe tags. Here are examples of how to do this safely:

C# Example:

```
StringBuilder sb = new StringBuilder(
HttpUtility.HtmlEncode(input));
sb.Replace("&lt;b&gt;", "<b>");
sb.Replace("&lt;/b&gt;", "<b>");
sb.Replace("&lt;i&gt;", "<i>");
sb.Replace("&lt;/i&gt;", "</i>");
Response.Write(sb.ToString());
```

VB.NET Example:

```
Dim sb As StringBuilder = New StringBuilder( _
HttpUtility.HtmlEncode(input))
sb.Replace("&lt;b&gt;", "<b>")
sb.Replace("&lt;/b&gt;", "<b>")
sb.Replace("&lt;i&gt;", "<i>")
sb.Replace("&lt;/i&gt;", "</i>")
Response.Write(sb.ToString())
```

Java Example:

```
public static String HTMLEncode(String aTagFragment){
final StringBuffer result = new StringBuffer();
final StringCharacterIterator iterator = new StringCharacterIterator(aTagFragment);
char character = iterator.current();
while (character != StringCharacterIterator.DONE ){
if (character = = '<') {
result.append("&lt;");
}
else if (character = = '>') {
result.append("&gt;");
}
else if (character = = '\"') {
result.append("&quot;");
}
else if (character = = '\') {
result.append("&#039;");
}
else if (character = = '\\') {
result.append("&#092;");
}
else if (character = = '&') {
result.append("&amp;");
}
else {
//the char is not a special one
//add it to the result as is
result.append(character);
}
character = iterator.next();
}
return result.toString();
}
```

The following recommendations will help you build web applications capable of withstanding Cross-Site Scripting attacks.

- Define what is allowed. Ensure that the web application validates all input parameters (cookies, headers, query strings, forms, hidden fields, etc.) against a stringent definition of expected results.
- Check the responses from POST and GET requests to ensure what is being returned is what is expected, and is valid.
- Remove conflicting characters, brackets, and single and double quotes from user input by encoding user supplied data. This will prevent inserted scripts from being sent to end users in a form that can be executed.
- Whenever possible, limit all client-supplied data to alphanumeric data. Using this filtering scheme, if a user entered "<script>alertdocumentcookie('aaa') </script>", it would be reduced to "scriptalertdocumentcookiescript". If non-alphanumeric characters must be used, encode them as HTML entities before using them in an HTTP response, so that they cannot be used to modify the structure of the HTML document.
- Use two-factor customer authentication mechanisms as opposed to single-factor authentication.
- Verify the origin of scripts before you modify or utilize them.

- Do not implicitly trust any script given to you by others (whether downloaded from the web, or given to you by an acquaintance) for use in your own code.

Most server side scripting languages provide built in methods to convert the value of the input variable into correct, non-interpretable HTML. These should be used to sanitize all input before displayed to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP / ASP.NET: Server.HTMLEncode (strHTML String)

For Security Operations:

Server-side encoding, where all dynamic content is first sent through an encoding function where Scripting tags will be replaced with codes in the selected character set, can help to prevent Cross-Site Scripting attacks. The drawback to server-side encoding is that it can be resource intensive, and may have a negative performance impact on some web servers.

If site users must be allowed to use HTML tags, such as a bulletin board where the user would be allowed to use formatting tags, limit the ones that can be used. Create a list of acceptable tags, such as bold, italic or underline, and only allow those to be used. Any other tags should be rejected. Below are a few regular expressions that will help detect Cross-Site Scripting.

Regex for a simple CSS attack:
/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/ix

The above regular expression would be added into a new Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII Cross-Site Scripting attempt"; flow:to_server,established;pcre:"/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/i";classtype:Web-application-attack; sid:9000; rev:5;)

Paranoid regex for CSS attacks:
/((\%3C)|<)[^\n]+((\%3E)|>)/I

This signature simply looks for the opening HTML tag, and its hex equivalent, followed by one or more characters other than the new line, and then followed by the closing tag or its hex equivalent. This may end up giving a few false positives depending upon how your Web application and Web server are structured, but it is guaranteed to catch anything that even remotely resembles a Cross-Site Scripting attack. From a public perspective, you can also strengthen educational programs to help consumers avoid online scams, such as phishing, that can be utilized in account hijackings and other forms of identify theft.

For QA:

Fixes for Cross-Site Scripting defects will ultimately require code based fixes. The steps detailed in the Developer and Security Operations section will provide any developer with the information necessary to remediate these issues. The following steps outline how to manually test an application for Cross-Site Scripting.

Step 1. Open any Web site in a browser, and look for places on the site that accept user input such as a search form or some kind of login page. Enter the word test in the search box and send this to the Web server.

Step 2. Look for the Web server to respond back with a page similar to something like "Your search for 'test' did not find any items" or "Invalid login test." If the word "test" appears in the results page, you are in luck.

Step 3. To test for Cross-Site Scripting, input the string "<script>alert('hello')</script>" without the quotes in the same search or login box you used before and send this to your Web server.

Step 4. If the server responds back with a popup box that says "hello", then the site is vulnerable to Cross-Site Scripting.

Step 5. If Step 4 fails and the Web site does not return this information, you still might be at risk. Click the 'View Source' option in your browser so you can see the actual HTML code of the Web page. Now find the <script> string that you sent the server. If you see the entire "<script>alert('hello')</script>" text in this source code, then the Web server is vulnerable to Cross-Site Scripting.

Reference:

ProCheckup:
Further .NET Request Validation Problems

Microsoft:
How to prevent Cross-Site Scripting security issues

OWASP:
Cross Site Scripting

HP:
Cross-Site Scripting Whitepaper

CERT:
CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests

OWASP:
Cross-Site Scripting

UK National Infrastructure Security Co-Ordination Centre (NISCC)
NISCC Vulnerability Advisory 165746/NISCC/DOTNET

Industry Reference Number(s)
Bugtraq ID: 20753

Attack Request:

```
GET /search.aspx?txtSearch="></XSS/*-*/STYLE=xss:e/**/xpression(alert(097531))> HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
```

Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou1nnj33b5w02x55ctl3kbed;amSessionId=037914641

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:54:42 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 7322



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Search Results
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Mutual, search, query,
find"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="default.aspx"
style="height:80px;width:183px;"><img src="images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
      <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="bank/login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
        <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
        <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
   </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">


<table cellspacing="0" width="100%">
  <tr>
    <td  ... {content removed}
```

High **Local File Inclusion/Reading Vulnerability**

File Names:
- http://demo.testfire.net:80/default.aspx?content=/..//..//..//..//..//..//..//..///boot.ini%00.htm

Summary: Severe vulnerabilities have been identified that would allow an attacker to remotely view the contents of files due to improper validation of input. The specific risks from exploitation depend upon the contents of the file being requested. Recommendations include adopting secure programming techniques to ensure that only expected data is accepted by an application.

Implication: An attacker can view the contents of various (possibly arbitrary) files on the system, which could potentially allow the attacker to recover application source code, system configuration information, or private data.

Fix:

For Development:

This problem arises from improper validation of characters accepted by the application. Any time a parameter is passed into a dynamically generated web page, it must be assumed that the data could be incorrectly formatted. The application should contain sufficient logic to handle the situation of a parameter not being passed in or being passed incorrectly. Keep in mind how the data is being submitted, as a result of a GET or a POST. Cookies should be treated the same as parameters when developing secure and stable code. The following recommendations will help to ensure you are delivering secure web applications.

- **Parameter not being passed:** If a parameter is expected to be passed to a dynamic web page and is omitted, the application should provide an acceptable error message to the user. Also, NEVER assume that a parameter is being passed before using it in an application.
- **Parameter of incorrect format:** A parameter should never be assumed to be of a valid format. This is especially true if the parameter is being passed to a SQL database. Any string that is passed directly to a database without first being checked for proper format can be a major security risk.

Also, just because a parameter is normally provided by a combo box or hidden field, DO NOT assume the format is correct. A hacker will try altering these parameters first if trying to break into your site.

- **Allowing file names to be passed in via a file name:** If a parameter is being used to determine which file to process in any way, NEVER allow the file name to be used before it is verified as valid. Specifically, you should test for the existence of characters that indicate directory traversal such as .../, c:\ and /.
- **Storing of critical data in hidden parameters:** Many programmers make the mistake of storing critical data in a hidden parameter or cookie. They assume that since the user doesn't see it, it's a good place to store data such as price, order number, etc. Both hidden parameters and cookies can be manipulated and returned to the server, so never assume the client returned what you set via a hidden parameter or cookie.

For Security Operations:

The specific fix for this vulnerability will need to be implemented in the actual script code. However, there are certain measures that can be initiated that will help in implementing a secure database protocol for your web application. Be advised each database has its own method of secure lock down.

- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- **Proper Error Handling:** Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

Ultimately, this problem will need to be rectified in the vulnerable script. If developed in-house, provide the developer with this report. If the script was downloaded from the Internet, or owned by a third party, please contact that vendor regarding the potential vulnerability and its proper mitigation.

Reference: **Input Validation Issues**
http://www.owasp.org/asac/input_validation/meta.shtml

Attack Request:

```
GET /default.aspx?content=/../..//../..//../..//../..//../..///boot.ini%00.htm HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k5
5pfimdwanvfujx2midkm45;amSessionId=037914644
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:42:36 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 7423



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
rel="stylesheet" type="text/css" /></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="default.aspx"
style="height:80px;width:183px;"><img src="images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="bank/login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
        <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
```

```
        <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
   </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">


<table cellspacing="0" width="100%">
  <tr>
    <td width="25%" class="bt br bb"><div id="Header1"><img ... {content removed}
```

High **HTTP Basic Logins Sent Over Unencrypted Connection**

File Names:
- http://demo.testfire.net:80/bank/members/

Summary: Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. http://demo.testfire.net:80/bank/members/ has failed this policy. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication: An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execition of actions not intended by the application developers.

Fix:

For Security Operations:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For Development:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For QA:

Test the application not only from the perspective of a normal user, but also from the perspective of a malicious one.

Attack Request:

```
GET /bank/members/ HTTP/1.1
Referer: http://demo.testfire.net:80/bank/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632;lang=
```

Attack Response:

```
HTTP/1.1 401 Unauthorized
Content-Length: 1656
Content-Type: text/html
Server: Microsoft-IIS/6.0
WWW-Authenticate: Basic realm="demo.testfire.net"
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:38:04 GMT

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<HTML><HEAD><TITLE>You are not authorized to view this page</TITLE>
<META HTTP-EQUIV="Content-Type" Content="text/html; charset=Windows-1252">
<STYLE type="text/css">
  BODY { font: 8pt/12pt verdana }
  H1 { font: 13pt/15pt verdana }
  H2 { font: 8pt/12pt verdana }
  A:link { color: red }
  A:visited { color: maroon }
</STYLE>
</HEAD><BODY><TABLE width=500 border=0 cellspacing=10><TR><TD>

<h1>You are not authorized to view this page</h1>
You do not have permission to view this directory or page using the credentials that you supplied because
your Web browser is sending a WWW-Authenticate header field that the Web server is not configured to
accept.
<hr>
<p>Please try the following:</p>
<ul>
<li>Contact the Web site administrator if you believe you should be able to view this directory or page.</li>
```

Click the Refresh button to try again with different credentials.

<h2>HTTP Error 401.2 - Unauthorized: Access is denied due to server configuration.
Internet Information Services (IIS)</h2>
<hr>
Technical Information (for support personnel)

Go to Microsoft Product Support Services and perform a title search for the words HTTP and 401.
Open IIS Help, which is accessible in IIS Manager (inetmgr),
and search for topics titled About Security, Authentication, and About Custom Error Messages.

</TD></TR></TABLE></BODY></HTML>

High **Unencrypted Login Form**

File Names:
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/admin/Login.aspx
- http://demo.testfire.net:80/bank/Login.aspx

Summary: An unencrypted login form has been discovered. Any area of a web application that possibly contains sensitive information or access to privileged functionality such as remote site administration functionality should utilize SSL or another form of encryption to prevent login information from being sniffed or otherwise intercepted or stolen. A page containing a login form should be SSL as well as the Action of the form. This will prevent Man-in-the-Middle attacks on the login form. Recommendations include ensuring that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

Implication: An attacker who exploited this design vulnerability would be able to utilize the information to escalate their method of attack, possibly leading to impersonation of a legitimate user, the theft of proprietary data, or execution of actions not intended by the application developers.

Fix:

For Security Operations:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For Development:

Ensure that sensitive areas of your web application have proper encryption protocols in place to prevent login information and other data that could be helpful to an attacker from being intercepted.

For QA:

Test the application not only from the perspective of a normal user, but also from the perspective of a malicious one.

Reference: **Advisory:**http://www.kb.cert.org/vuls/id/466433

Attack Request:

```
GET /bank/login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:37:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8729
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Mutual
Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
```

```
 <form id="frmSearch" method="get" action="/search.aspx">
  <table width="100%" border="0" cellpadding="0" cellspacing="0">
   <tr>
      <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
    <td align="right" valign="top">
     <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
       <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
       <input type="submit" value="Go" />
    </td>
   </tr>
   <tr>
    <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
   </tr>
  </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">

 ... {content removed}
```

High **Microsoft ASP.NET 2.0 Request Filtering Bypass Cross-Site Scripting Vulnerability**

File Names:
- http://demo.testfire.net:80/bank/login.aspx

Summary: A Cross-Site Scripting vulnerability has been detected in Microsoft ASP.NET request filtering. Web applications that are coded in any .NET language and rely only on the default .NET request filtering are vulnerable to Cross-Site Scripting. If exploited, an attacker can manipulate or steal cookies, create requests that can be mistaken for those of a valid user, compromise confidential information, or execute malicious code on end user systems. Recommendations include implementing secure programming techniques that ensure proper filtration of user-supplied data, and encoding all user-supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Execution: The following is a proof of concept:

http://www.example.com/MyApp.aspx?myvar=<~/XSS/*-*/STYLE=xss:e/**/xpression(alert('XSS'))>

Implication: Cross-Site Scripting happens when user input from a web client is immediately included via server-side scripts in a dynamically generated web page. Via social engineering, an attacker can trick a victim, such as through a malicious link or "rigged" form, to submit information which will be altered to include attack code and then sent to the legitimate server. The injected code is then reflected back to the user's browser which executes it because it came from a trusted server.

The main problems associated with successful Cross-Site Scripting attacks are:

- Account hijacking
- Javascript-based worm propagation
- Information theft
- Denial of service
- Browser redirection
- Manipulation of user settings

For more detailed information on Cross-Site Scripting attacks, see the HP Application Security Center Cross-Site Scripting whitepaper.

Vulnerable products and versions:

- Microsoft Windows Server 2003 Standard Edition Build 3790.srv03_sp1_rtm.050324-1447 Service Pack 1
- Microsoft IIS 6.0
- Microsoft ASP .NET Framework Version 2.0.50727.42
- Microsoft Internet Explorer 6.0.2900.2180.xpsp_sp2_gdr.050301-1519
- Microsoft Internet Explorer 7.0.5450.4 Beta 3
- Microsoft Internet Explorer 7.0.5730.11

Note: We were unable to confirm the vulnerability in the latest versions of Microsoft ASP .NET Framework and Microsoft Internet Explorer.

Fix: **For Development:**

Cross-Site Scripting attacks can be avoided by carefully validating all input, and properly encoding all output. Do not rely solely on default ASP.NET validation controls. In addition to using the default validation controls, properly sanitize all input parameters on server side applications by adopting a whitelist strategy to input validation. You may also validate input parameters directly in your code. Always use as strict a pattern as you can possibly allow.

Encoding of output ensures that any scriptable content is properly encoded for HTML before being sent to the client. This is done with the function HttpUtility.HtmlEncode, as shown in the following Label control sample:

Label2.Text = HttpUtility.HtmlEncode(input)

Be sure to consider all paths that user input takes through your application. For instance, if data is entered by the user, stored in a database, and then redisplayed later, you must make sure it is properly encoded each time it is retrieved. If you must allow free-format text input, such as in a message board, and you wish

to allow some HTML formatting to be used, you can handle this safely by explicitly allowing only a small list of safe tags. Here are examples of how to do this safely:

C# Example:

```
StringBuilder sb = new StringBuilder(
HttpUtility.HtmlEncode(input));
sb.Replace("&lt;b&gt;", "<b>");
sb.Replace("&lt;/b&gt;", "<b>");
sb.Replace("&lt;i&gt;", "<i>");
sb.Replace("&lt;/i&gt;", "</i>");
Response.Write(sb.ToString());
```

VB.NET Example:

```
Dim sb As StringBuilder = New StringBuilder( _
HttpUtility.HtmlEncode(input))
sb.Replace("&lt;b&gt;", "<b>")
sb.Replace("&lt;/b&gt;", "<b>")
sb.Replace("&lt;i&gt;", "<i>")
sb.Replace("&lt;/i&gt;", "</i>")
Response.Write(sb.ToString())
```

Java Example:

```
public static String HTMLEncode(String aTagFragment){
final StringBuffer result = new StringBuffer();
final StringCharacterIterator iterator = new StringCharacterIterator(aTagFragment);
char character = iterator.current();
while (character != StringCharacterIterator.DONE ){
if (character = = '<') {
result.append("&lt;");
}
else if (character = = '>') {
result.append("&gt;");
}
else if (character = = '\"') {
result.append("&quot;");
}
else if (character = = '\') {
result.append("&#039;");
}
else if (character = = '\\') {
result.append("&#092;");
}
else if (character = = '&') {
result.append("&amp;");
}
else {
//the char is not a special one
//add it to the result as is
result.append(character);
}
character = iterator.next();
}
return result.toString();
}
```

The following recommendations will help you build web applications capable of withstanding Cross-Site Scripting attacks.

- Define what is allowed. Ensure that the web application validates all input parameters (cookies, headers, query strings, forms, hidden fields, etc.) against a stringent definition of expected results.
- Check the responses from POST and GET requests to ensure what is being returned is what is expected, and is valid.
- Remove conflicting characters, brackets, and single and double quotes from user input by encoding user supplied data. This will prevent inserted scripts from being sent to end users in a form that can be executed.
- Whenever possible, limit all client-supplied data to alphanumeric data. Using this filtering scheme, if a user entered "<script>alertdocumentcookie('aaa') </script>", it would be reduced to "scriptalertdocumentcookiescript". If non-alphanumeric characters must be used, encode them as HTML entities before using them in an HTTP response, so that they cannot be used to modify the structure of the HTML document.
- Use two-factor customer authentication mechanisms as opposed to single-factor authentication.
- Verify the origin of scripts before you modify or utilize them.
- Do not implicitly trust any script given to you by others (whether downloaded from the web, or given to you by an acquaintance) for use in your own code.

Most server side scripting languages provide built in methods to convert the value of the input variable into correct, non-interpretable HTML. These should be used to sanitize all input before displayed to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP / ASP.NET: Server.HTMLEncode (strHTML String)

For Security Operations:

Server-side encoding, where all dynamic content is first sent through an encoding function where Scripting tags will be replaced with codes in the selected character set, can help to prevent Cross-Site Scripting attacks. The drawback to server-side encoding is that it can be resource intensive, and may have a negative performance impact on some web servers.

If site users must be allowed to use HTML tags, such as a bulletin board where the user would be allowed to use formatting tags, limit the ones that can be used. Create a list of acceptable tags, such as bold, italic or underline, and only allow those to be used. Any other tags should be rejected. Below are a few regular expressions that will help detect Cross-Site Scripting.

Regex for a simple CSS attack:
/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/ix

The above regular expression would be added into a new Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NII Cross-Site Scripting attempt"; flow:to_server,established;pcre:"/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/i";classtype:Web-application-attack; sid:9000; rev:5;)

Paranoid regex for CSS attacks:
/((\%3C)|<)[^\n]+((\%3E)|>)/I

This signature simply looks for the opening HTML tag, and its hex equivalent, followed by one or more characters other than the new line, and then followed by the closing tag or its hex equivalent. This may end up giving a few false positives depending upon how your Web application and Web server are structured, but it is guaranteed to catch anything that even remotely resembles a Cross-Site Scripting attack. From a public perspective, you can also strengthen educational programs to help consumers avoid online scams, such as phishing, that can be utilized in account hijackings and other forms of identify theft.

For QA:

Fixes for Cross-Site Scripting defects will ultimately require code based fixes. The steps detailed in the Developer and Security Operations section will provide any developer with the information necessary to remediate these issues. The following steps outline how to manually test an application for Cross-Site Scripting.

Step 1. Open any Web site in a browser, and look for places on the site that accept user input such as a search form or some kind of login page. Enter the word test in the search box and send this to the Web server.

Step 2. Look for the Web server to respond back with a page similar to something like "Your search for 'test' did not find any items" or "Invalid login test." If the word "test" appears in the results page, you are in luck.

Step 3. To test for Cross-Site Scripting, input the string "<script>alert('hello')</script>" without the quotes in the same search or login box you used before and send this to your Web server.

Step 4. If the server responds back with a popup box that says "hello", then the site is vulnerable to Cross-Site Scripting.

Step 5. If Step 4 fails and the Web site does not return this information, you still might be at risk. Click the 'View Source' option in your browser so you can see the actual HTML code of the Web page. Now find the <script> string that you sent the server. If you see the entire "<script>alert('hello')</script>" text in this source code, then the Web server is vulnerable to Cross-Site Scripting.

Reference:

ProCheckup:
Further .NET Request Validation Problems

Microsoft:
How to prevent Cross-Site Scripting security issues

OWASP:
Cross Site Scripting

HP:
Cross-Site Scripting Whitepaper

CERT:
CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests

OWASP:
Cross-Site Scripting

UK National Infrastructure Security Co-Ordination Centre (NISCC)
NISCC Vulnerability Advisory 165746/NISCC/DOTNET

Industry Reference Number(s)
Bugtraq ID: 20753

Attack Request:

```
POST /bank/login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 83
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
```

1nnj33b5w02x55ctl3kbed;amSessionId=037914641

uid=" <~/XSS/*-*/STYLE=xss:e/**/xpression(alert(097531))>&passw=foo&btnSubmit=Login

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:54:57 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8874



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Mutual
Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
      <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
        <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
        <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
   </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">

 ... {content removed}
```

Medium **Directory Listing**

File Names:
- http://demo.testfire.net:80/pr/
- http://demo.testfire.net:80/bank/

Summary: A serious Directory Listing vulnerability was discovered within your web application. Risks associated with an attacker discovering a Directory Listing, which is a complete index of all of the resources located in that directory, result from the fact that files that should remain hidden, such as data files, backed-up source code, or applications in development, may then be visible. The specific risks depend upon the specific files that are listed and accessible. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that could expose private files and provide information that could be utilized by an attacker when formulating or conducting an attack.

Execution: http://demo.testfire.net:80/pr/

Implication: Risks associated with an attacker discovering a Directory Listing on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat from an accessible Directory Listing is that hidden files such as data files, source code, or applications under development will then be visible to a potential attacker. In addition to accessing files containing sensitive information, other risks include an attacker utilizing the information discovered in that directory to perform other types of attacks.

Fix: **For Development:** Unless you are actively involved with implementing the web application server, there is not a wide range of available solutions to prevent problems that can occur from an attacker finding a Directory Listing. Primarily, this problem will be resolved by the web application server administrator. However, there are certain actions you can take that will help to secure your web application.

- Restrict access to important files or directories only to those who actually need it.

- Ensure that files containing sensitive information are not left publicly accessible, or that comments left inside files do not reveal the locations of directories best left confidential.

For Security Operations:

One of the most important aspects of web application security is to restrict access to important files or directories only to those individuals who actually need to access them. Ensure that the private architectural structure of your web application is not exposed to anyone who wishes to view it as even seemingly innocuous directories can provide important information to a potential attacker.

The following recommendations can help to ensure that you are not unintentionally allowing access to either information that could be utilized in conducting an attack or propriety data stored in publicly accessible directories.

- Turn off the Automatic Directory Listing feature in whatever application server package that you utilize.
- Restrict access to important files or directories only to those who actually need it.
- Ensure that files containing sensitive information are not left publicly accessible.
- Don't follow standard naming procedures for hidden directories. For example, don't create a hidden directory called "cgi" that contains cgi scripts. Obvious directory names are just that...readily guessed by an attacker.

Remember, the harder you make it for an attacker to access information about your web application, the more likely it is that he will simply find an easier target.

For QA:

For reasons of security, it is important to test the web application not only from the perspective of a normal user, but also from that of a malicious one. Whenever possible, adopt the mindset of an attacker when testing your web application for security defects. Access your web application from outside your firewall or IDS. Utilize Google or another search engine to ensure that searches for vulnerable files do not return information from regarding your web application. For example, an attacker will utilize a search engine, and search for directory listings such as the following: "index of / cgi-bin". Make sure that your directory structure is not obvious, and that only files that are necessary are capable of being accessed.

Reference:

Apache:

Security Tips for Server Configuration

Protecting Confidential Documents at Your Site

Securing Apache - Access Control

IIS:

Implementing NTFS Standard Permissions on Your Web Site

Netscape:

Controlling Access to Your Server

General:

Password-protecting web pages

Web Security

Attack Request:

```
GET /pr/ HTTP/1.1
Referer: http://demo.testfire.net:80/pr/communityannualreport.pdf
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k5
5pfimdwanvfujx2midkm45;amSessionId=037914644
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Length: 584
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:45:14 GMT

<html><head><META http-equiv="Content-Type" content="text/html;
charset=UTF-8"><title>demo.testfire.net - /pr/</title></head><body><H1>demo.testfire.net -
/pr/</H1><hr>

<pre><A HREF="/">[To Parent Directory]</A><br><br> 7/20/2007  7:45 AM       63887 <A
HREF="/pr/communityannualreport.pdf">communityannualreport.pdf</A><br> 11/1/2006  9:13 PM
804 <A HREF="/pr/Docs.xml">Docs.xml</A><br> 3/13/2006  9:12 AM       11281 <A
HREF="/pr/Draft.rtf">Draft.rtf</A><br> 7/20/2007  8:41 AM      187754 <A
HREF="/pr/Q3_earnings.rtf">Q3_earnings.rtf</A><br></pre><hr></body></html>
```

Medium

IIS Missing Host Header Internal IP Address Disclosure

File Names:

- http://demo.testfire.net:80/admin

Summary: In certain configurations, IIS may disclose its internal IP address when a HTTP/1.0 request without a Host header.

Execution: The following example HTTP request assumes that there is a directory called 'images' on the server:

GET /images HTTP/1.0

Examine the 'Location' header of the HTTP response for an internal IP address.

Implication: Information disclosure vulnerabilities reveal sensitive information about a system or web application to an attacker. An attacker can use this information to learn more about a system when attempting to gain unauthorized access.

Fix: Apply the configuration changes described in Microsoft Knowledge Base article Q218180.

Reference: **Microsoft Knowledge Base Article Q218180**
http://support.microsoft.com/default.aspx?scid=KB;EN-US;Q218180

Bugtraq
http://www.securityfocus.com/bid/3159/

Attack Request:
```
GET /admin HTTP/1.0
User-Agent: WebInspect
```

Attack Response:
```
HTTP/1.1 301 Moved Permanently
Content-Length: 150
Content-Type: text/html
Location: http://192.168.1.117/admin/
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:55:02 GMT
Connection: close

<head><title>Document Moved</title></head>
<body><h1>Object Moved</h1>This document may be found <a
HREF="http://192.168.1.117/admin/">here</a></body>
```

Medium **Common Application Test Files**

File Names:
- http://demo.testfire.net:80/test.aspx

Summary: Testing-related application pages were found. Test pages are usually implemented ad-hoc and often do not adhere to the security requirements/guidelines of the rest of the application, making them a potential security hazard. Recommendations include restricting access to only those with an actual need to access the page, or if applicable, removing the information from the production server.

Implication: The impact of this vulnerability depends on what information/functionality was discovered in the test-related application page.

Fix: **For Security Operations:**
Either remove this file from the production server or restrict access to only authorized users. Restrict access to important files or directories only to those who actually need it. Enforce consistent authentication across your entire application and apply it to the entire directory structure, including subdirectories.

For Development:
Ensure that files containing sensitive information or test functionality are not left publicly accessible.

For QA:
Ensure that files containing sensitive information or test functionality are not left publicly accessible.

Reference: **Implementing Basic Authentication in IIS:**
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache:
http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:
```
GET /test.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k55pfimdwanvfujx2midkm45;amSessionId=037914644
```

Attack Response:
```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:41:11 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 558


<html>
<head><title>
Altoro Mutual Test Page
</title>
</head>

 <body><center>
<img src="images/logo.gif">
```

```
<p>
<br>
If ASP.Net is installed correctly a message should appear below.

<p>


<p>
<span style='color: red;'>
ASP.Net is installed and functioning
</span>

</p>
<p>
<br>
If nothing appears above in red, you do not have ASP.Net installed correctly.</p><p>  Please refer to the documentation provided by Microsoft to get ASP.Net installed and functioning.</P>
</center>

</body>
</html>
```

Low **Possible Server Path Disclosure (win32)**

File Names:

- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/queryxpath.aspx
- http://demo.testfire.net:80/bank/customize.aspx
- http://demo.testfire.net:80/feedback.aspx
- http://demo.testfire.net:80/comment.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx

Summary: A minor vulnerability has been detected within your web application due to the discovery of a fully qualified path name to the root of your system. This most often occurs in context of an error being produced by the web application. Fully qualified server path names allow an attacker to know the file system structure of the web server, which is a baseline for many other types of attacks to be successful. Recommendations include adopting a consistent error handling scheme and mechanism that prevents fully qualified path names from being displayed.

Fix:

For Development:

Don't display fully qualified pathnames as part of error or informational messages. At the least, fully qualified pathnames can provide an attacker with important information about the architecture of web application.

For Security Operations:

The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? It is often a seemingly innocuous piece of information that provides an attacker with the means to discover something else which he can then utilize when conducting an attack.

Attack Request:

```
POST /bank/login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 33
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
```

Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k5
5pfimdwanvfujx2midkm45;amSessionId=037914644

uid=%00&passw=foo&btnSubmit=Login

Attack Response:

HTTP/1.1 500 Internal Server Error
Connection: close
Date: Wed, 04 Feb 2009 06:43:42 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 5011

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_head"><title>
Altoro Mutual: Server Error
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /></head>
<body style="margin-top:5px;">

<form id="frmSearch" method="get" action="/search.aspx">
<table width="100%" border="0" cellpadding="0" cellspacing="0">
<tr>
<td rowspan="2"><a id="_ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"></td>
<td align="right" valign="top">
<a id="_ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In | <a
id="_ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us | <a
id="_ctl0_HyperLink4" href="../feedback.aspx">Feedback | <label for="txtSearch">Search</label>
<input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
<input type="submit" value="Go" />
</td>
</tr>
<tr>
<td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
</tr>
</table>
</form>

<h1>An Error Has Occurred</h1>

< ... {content removed}

Low **Internal IP Disclosure**

File Names:

- http://demo.testfire.net:80/admin

Summary:

A string matching an internal/reserved IPv4 or IPv6 address range was discovered. This may disclose information about the IP addressing scheme of the internal network and can be valuable to attackers.
Internal IPv4/IPv6 ranges are:
10.x.x.x
172.16.x.x through 172.31.x.x
192.168.x.x
fd00::x
If not a part of techical documentation, recommendations include removing the string from the production server.

Fix:

This issue can appear for several reasons. The most common is that the application or webserver error message discloses the IP address. This can be solved by determining where to turn off detailed error messages in the application or webserver. Another common reason is due to a comment located in the source of the webpage. This can easily be removed from the source of the page.

Attack Request:
GET /admin HTTP/1.0
User-Agent: WebInspect

Attack Response:
HTTP/1.1 301 Moved Permanently
Content-Length: 150
Content-Type: text/html
Location: http://192.168.1.117/admin/
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:55:02 GMT
Connection: close

<head><title>Document Moved</title></head>
<body><h1>Object Moved</h1>This document may be found <a
HREF="http://192.168.1.117/admin/">here</body>

Low **Server Error Response**

File Names:

- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%00
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%0aA:B
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%2500
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%250a
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=/%2A
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%2A
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=/
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%2f%2c%25ENV%2c%2f
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=%0d%0aInjectedHeader:%20InjectedValue
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=())[]]{}}
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=,`@^*$;#
- http://demo.testfire.net:80/bank/ws.asmx?WSDL="
- http://demo.testfire.net:80/bank/ws.asmx?WSDL='
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- http://demo.testfire.net:80/bank/ws.asmx?WSDL=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%00
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%0aA:B
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%2500
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%250a
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=/%2A
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%2A
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=/
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%2f%2c%25ENV%2c%2f
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=%0d%0aInjectedHeader:%20InjectedValue
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

- http://demo.testfire.net:80/bank/ws.asmx?wsdl=())[]]{}}
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=,`@^*$;#
- http://demo.testfire.net:80/bank/ws.asmx?wsdl="
- http://demo.testfire.net:80/bank/ws.asmx?wsdl='
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA
- http://demo.testfire.net:80/bank/ws.asmx?wsdl=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/comment.aspx
- http://demo.testfire.net:80/bank/customize.aspx
- http://demo.testfire.net:80/bank/queryxpath.aspx
- http://demo.testfire.net:80/bank/login.aspx

Summary: A server error response was detected. The server could be experiencing errors due to a misbehaving application, a misconfiguration, or a malicious value sent during the auditing process. While error responses in and of themselves are not dangerous, per se, the error responses give attackers insight into how the application handles error conditions. Errors that can be remotely triggered by an attacker can also potentially lead to a denial of service attack or other more severe vulnerability. Recommendations include designing and adding consistent error handling mechanisms which are capable of handling any user input to your web application, providing meaningful detail to end-users, and preventing error messages that might provide information useful to an attacker from being displayed.

Implication: The server has issued a 500 error response. While the body content of the error page may not expose any information about the technical error, the fact that an error occurred is confirmed by the 500 status code. Knowing whether certain inputs trigger a server error can aid or inform an attacker of potential vulnerabilities.

Fix:

For Security Operations:

Server error messages, such as "File Protected Against Access", often reveal more information than intended. For instance, an attacker who receives this message can be relatively certain that file exists, which might give him the information he needs to pursue other leads, or to perform an actual exploit. The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any server error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

Removing Detailed Error Messages

Find instructions for turning off detailed error messaging in IIS at this link:

http://support.microsoft.com/kb/294807

For Development:

From a development perspective, the best method of preventing problems from arising from server error messages is to adopt secure programming techniques that prevent problems that might arise from an attacker discovering too much information about the architecture and design of your web application. The following recommendations can be used as a basis for that.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad. Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

For QA:

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? Inconsistent methods of dealing with errors gives an attacker a very powerful way of gathering information about your web application.

Reference:

Apache:

Security Tips for Server Configuration

Protecting Confidential Documents at Your Site

Securing Apache - Access Control

Microsoft:

How to set required NTFS permissions and user rights for an IIS 5.0 Web server

Default permissions and user rights for IIS 6.0

Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes

Attack Request:

```
POST /subscribe.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/subscribe.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 2129
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641

txtEmail=AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ... {content removed}
```

Attack Response:

```
HTTP/1.1 500 Internal Server Error
Connection: close
Date: Wed, 04 Feb 2009 06:46:39 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 4259


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_head"><title>
Altoro Mutual: Server Error
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css" rel="stylesheet" type="text/css" /></head>
<body style="margin-top:5px;">

<form id="frmSearch" method="get" action="/search.aspx">
<table width="100%" border="0" cellpadding="0" cellspacing="0">
<tr>
<td rowspan="2"></td>
<td align="right" valign="top">
Sign In | Contact Us | Feedback | <label for="txtSearch">Search</label>
<input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
<input type="submit" value="Go" />
</td>
</tr>
<tr>
<td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"></td>
</tr>
</table>
</form>

<h1>An Error Has Occurred</h1>

<h2>Summary:</ ... {content removed}

Low **Microsoft ASP.NET Debugging Enabled**

File Names:

- http://demo.testfire.net:80/bank/account.aspx
- http://demo.testfire.net:80/bank/apply.aspx
- http://demo.testfire.net:80/bank/customize.aspx
- http://demo.testfire.net:80/bank/main.aspx
- http://demo.testfire.net:80/bank/queryxpath.aspx
- http://demo.testfire.net:80/bank/servererror.aspx
- http://demo.testfire.net:80/bank/transaction.aspx
- http://demo.testfire.net:80/bank/transfer.aspx
- http://demo.testfire.net:80/servererror.aspx?aspxerrorpath=/bank/account.aspx.cs
- http://demo.testfire.net:80/Default.aspx
- http://demo.testfire.net:80/badfile123.aspx
- http://demo.testfire.net:80/admin/Login.aspx
- http://demo.testfire.net:80/bank/Login.aspx
- http://demo.testfire.net:80/comment.aspx
- http://demo.testfire.net:80/search.aspx
- http://demo.testfire.net:80/default.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/feedback.aspx
- http://demo.testfire.net:80/survey_questions.aspx
- http://demo.testfire.net:80/subscribe.aspx

Summary: Microsoft ASP.NET debugging was found to be enabled. With debugging enabled, the debug command can be executed on the application remotely. If exploited, an attacker could learn sensitive information about the target system and the target Web application. Recommendations include modifying the web.config file to disable debugging.

Execution: A custom HTTP verb exists which allows a remote user to enable debugging support in ASP.NET. If the verb 'DEBUG' is sent, the debug handler is loaded in place of the URL that was requested. If debugging is enabled, ASP.NET looks for a header called 'Command' with one of the following values: stop-debug or start-debug.

By sending an HTTP Request as shown below, we get the following response indicating that remote debugging is possible.

HTTP Request:

DEBUG /CommandExecution.aspx HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Command: stop-debug
Content-Length: 0

HTTP Response:

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
X-Powered-By: ASP.NET
X-AspNet-Version: 1.1.4322
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2

OK

Implication: Depending upon the access control present on the server, a remote debug session can potentially disclose sensitive information about the target system as well as information about the target web application. The attacker could then use this information to launch further attacks against the affected host.

Fix:

For Security Operations:

Disable debugging for any vulnerable directory. To disable debugging, edit the web.config file as follows:

<configuration><system.web><compilation debug=false>

For Developers:

Ultimately, debugging must be disabled by Security Operations before implementing the Web application in a production environment.

For QA:

For security reasons, it is important to test the web application not only from the perspective of a normal user, but also from that of a malicious one. Whenever possible, adopt the mindset of an attacker when testing your web application for security defects.

Attempt to send an HTTP Request as shown in the Execution section. If the response indicates that remote debugging is possible, report your findings to Security Operations.

Reference:

Microsoft

Debugging ASP.NET Web Applications

Advisory

HOW TO: Disable Debugging for ASP.NET Applications

Attack Request:

DEBUG /bank/account.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Command: stop-debug
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k55pfimdwanvfujx2midkm45;amSessionId=037914644

Attack Response:

HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:55:37 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private
Content-Type: text/html; charset=utf-8
Content-Length: 2

OK

Low **Login Interface (login.aspx)**

File Names:

- http://demo.testfire.net:80/bank/Login.aspx
- http://demo.testfire.net:80/admin/Login.aspx

Summary: The predictably named file /bank/Login.aspx was found. Recommendations include evaluating whether this file should be publicly accessible.

Implication: The implications of this file depend on what it contains. Often, it is a seemingly innocuous piece of information that can complete the knowledge a potential attacker needs to compromise a site.

Fix:

For Security Operations:

Evaluate whether the presence of the discovered file is intended and should be directly accessible to remote users. Restrict access to important directories or files.

For Development:

Make sure the naming conventions of your file system do not reveal information of value to a potential attacker.

For QA:

Check to make sure things of value to a potential attacker have not been left publicly available.

Reference:

Apache:

Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

IIS:

Implementing NTFS Standard Permissions on Your Web Site

Netscape:
Controlling Access to Your Server

General:
Password-protecting web pages
Web Security

Attack Request:
```
GET /bank/Login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641
```

Attack Response:
```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:47:56 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8729



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Online Banking Login
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Mutual
Login, login, authenticate"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
         <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
         <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
   </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">

 ... {content removed}
```

Low **Administrative Directories**

File Names:
- http://demo.testfire.net:80/admin/

Summary:
Administrative directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering an administrative directory on your application server typically include the potential for the attacker to use the administrative applications to affect the operations of the web site. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Implication: The primary danger from an attacker finding a publicly available directory on your web application server depends on what type of directory it is, and what files it contains. Administrative directories typically contain applications capable of changing the configuration of the running software; an attacker who gains access to an administrative application can drastically affect the operation of the web site.

Fix:

For Security Operations:

You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:

This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:

This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache

http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

```
GET /admin/ HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou1nnj33b5w02x55ctl3kbed;amSessionId=037914641
```

Attack Response:

```
HTTP/1.1 403 Forbidden
Content-Length: 218
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:37:15 GMT

<html><head><title>Error</title></head><body><head><title>Directory Listing Denied</title></head>
<body><h1>Directory Listing Denied</h1>This Virtual Directory does not allow contents to be listed.</body></body></html>
```

Low

Common Web Site Structure Directories

File Names:

- http://demo.testfire.net:80/images/

Summary: Directory Enumeration vulnerabilities were discovered within your web application. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:

You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:

This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:

This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache

http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

```
GET /images/ HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
```

Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k55pfimdwanvfujx2midkm45;amSessionId=037914644

Attack Response:

HTTP/1.1 403 Forbidden
Content-Length: 218
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:37:31 GMT

<html><head><title>Error</title></head><body><head><title>Directory Listing Denied</title></head><body><h1>Directory Listing Denied</h1>This Virtual Directory does not allow contents to be listed.</body></body></html>

Low

E-Commerce and Financial Directories

File Names:

- http://demo.testfire.net:80/bank/

Summary:

E-Commerce and/or financial-related directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:

You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:

This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:

This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Implementing Basic Authentication in Apache

http://httpd.apache.org/docs/howto/auth.html#intro

Attack Request:

GET /bank/ HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou1nnj33b5w02x55ctl3kbed;amSessionId=037914641

Attack Response:

HTTP/1.1 200 OK
Content-Length: 2364
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:38:00 GMT

<html><head><META http-equiv="Content-Type" content="text/html; charset=UTF-8"><title>demo.testfire.net - /bank/</title></head><body><H1>demo.testfire.net - /bank/</H1><hr>

<pre><A HREF="/">[To Parent Directory]</A>

 1/22/2007 4:17 PM <dir> <A HREF="/bank/20060308_bak/">20060308_bak</A>
11/20/2006 9:05 AM 1831 <A HREF="/bank/account.aspx">account.aspx</A>
 1/9/2008 10:00 AM 4277 <A HREF="/bank/account.aspx.cs">account.aspx.cs</A>
11/20/2006 9:05 AM 771 <A HREF="/bank/apply.aspx">apply.aspx</A>
11/20/2006 9:05 AM 2828 <A HREF="/bank/apply.aspx.cs">apply.aspx.cs</A>
11/10/2006 12:20 PM 2236 <A HREF="/bank/bank.master">bank.master</A>
 7/16/2007 7:35 AM 1134 <A HREF="/bank/bank.master.cs">bank.master.cs</A>
11/20/2006 9:05 AM 904 <A HREF="/bank/customize.aspx">customize.aspx</A>
11/20/2006 9:05 AM 1955 <A HREF="/bank/customize.aspx.cs">customize.aspx.cs</A>
 7/23/2007 3:26 PM 1806 <A

HREF="/bank/login.aspx">login.aspx</A>
 7/23/2007 3:27 PM 5847 <A
HREF="/bank/login.aspx.cs">login.aspx.cs</A>
 11/1/2006 7:42 PM 78 <A
HREF="/bank/logout.aspx">logout.aspx</A>
 1/3/2008 11:01 AM 3361 <A
HREF="/bank/logout.aspx.cs">logout.aspx.cs</A>
 7/16/2007 7:21 AM 935 <A
HREF="/bank/main.aspx">main.aspx</A>
 7/16/2007 8:36 AM 3951 <A
HREF="/bank/main.aspx.cs">main.aspx.cs</A>
 10/2/2006 9:21 AM <dir> <A
HREF="/bank/members/">members</A>
 1/12/2007 12:55 PM 1414 <A
HREF="/bank/mozxpath.js">mozxpath.js</A>
11/20/2006 9:05 AM 785 <A
HREF="/bank/queryxpath.aspx">queryxpath.aspx</A>
11/20/2006 9:05 AM 1838 <A
HREF="/bank/queryxpath.aspx.cs">queryxpath.aspx.cs</A>
 7/18/2007 4:13 PM 499 <A
HREF="/bank/servererror.aspx">servererror ... {content removed}

Low **IIS/Microsoft Directories**

File Names:
- http://demo.testfire.net:80/aspnet_client/

Summary: IIS/Microsoft product directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:

You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:

This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:

This problem will be resolved by the web application server administrator.

Reference: **Implementing Basic Authentication in IIS**

http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Attack Request:

```
GET /aspnet_client/ HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=k55pfimdwanvfujx2midkm45;amSessionId=037914644
```

Attack Response:

```
HTTP/1.1 403 Forbidden
Content-Length: 218
Content-Type: text/html
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:53:37 GMT

<html><head><title>Error</title></head><body><head><title>Directory Listing Denied</title></head>
<body><h1>Directory Listing Denied</h1>This Virtual Directory does not allow contents to be
listed.</body></body></html>
```

Low **Possible Insecure Cryptographic Hash (MD Family)**

File Names:
- http://demo.testfire.net:80/pr/Q3_earnings.rtf

Summary: A string of hexadecimal digits matching the length of a cryptographic hash from the MD family was detected. Cryptographic hashes are often used to protect passwords, session information, and other sensitive data. There are multiple hashing algorithms in the MD family. By far the most commonly used algorithm is MD5, though MD4 and MD2 are still used with various public key and digital certificate systems. There are known attacks against MD5, MD4, and MD2. These hashes are also susceptible to Rainbow table attacks unless the input is properly salted. As such the MD family of cryptographic hashing functions should not be considered secure and should only be used in certain situations.

Implication: Hashes produced by the MD family should only be used for short-lived uses where the hash and/or hashed data is not highly security sensitive, or for uses were uniqueness is not a critical requirement. MD Hashes should not be used for any type of long term application such as verifying the integrity of a file or for password storage.

Fix:

For Development:

The application should only use cryptographically secure hashing algorithms, such as SHA-224, SHA-256, SHA-384, or SHA-512. Hashes representing sensitive data should be salted to reduce the effectiveness of rainbow tables.

For Security Operations:

Implement a security policy that precludes the use of MD5, MD4, or MD2 for cryptographic functionality.

For QA:

Make sure that the application is not relying on MD5, MD4, or MD2 for cryptographic functionality.

Reference:

MD5
http://en.wikipedia.org/wiki/MD5

Cryptographic Salting
http://en.wikipedia.org/wiki/Salt_%28cryptography%29

Project Rainbow Crack
http://www.antsight.com/zsl/rainbowcrack/

Attack Request:

```
GET /pr/Q3_earnings.rtf HTTP/1.1
Referer: http://demo.testfire.net:80/pr/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632;lang=
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Length: 187754
Content-Type: application/rtf
Last-Modified: Fri, 20 Jul 2007 14:41:48 GMT
Accept-Ranges: bytes
ETag: "01eee19dccac71:5ae"
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:45:15 GMT

{\rtf1\adeflang1025\ansi\ansicpg1252\uc1\adeff0\deff0\stshfdbch0\stshfloch0\stshfhich0\stshfbi0\deflang1
033\deflangfe1033{\fonttbl{\f0\froman\fcharset0\fprq2{\*\panose 02020603050405020304}Times New
Roman;}{\f1\fswiss\fcharset0\fprq2{\*\panose 020b0604020202020204}Arial;}
{\f2\fmodern\fcharset0\fprq1{\*\panose 02070309020205020404}Courier
New;}{\f3\froman\fcharset2\fprq2{\*\panose
05050102010706020507}Symbol;}{\f10\fnil\fcharset2\fprq2{\*\panose
05000000000000000000}Wingdings;}
{\f88\fswiss\fcharset0\fprq2{\*\panose
020b0604030504040204}Verdana;}{\f261\froman\fcharset238\fprq2 Times New Roman
CE;}{\f262\froman\fcharset204\fprq2 Times New Roman Cyr;}{\f264\froman\fcharset161\fprq2 Times New
 Roman Greek;}
{\f265\froman\fcharset162\fprq2 Times New Roman Tur;}{\f266\fbidi \froman\fcharset177\fprq2 Times
New Roman (Hebrew);}{\f267\fbidi \froman\fcharset178\fprq2 Times New Roman
(Arabic);}{\f268\froman\fcharset186\fprq2 Times New Roman Baltic;}
{\f269\froman\fcharset163\fprq2 Times New Roman (Vietnamese);}{\f271\fswiss\fcharset238\fprq2 Arial
CE;}{\f272\fswiss\fcharset204\fprq2 Arial Cyr;}{\f274\fswiss\fcharset161\fprq2 Arial
Greek;}{\f275\fswiss\fcharset162\fprq2 Arial Tur;}
{\f276\fbidi \fswiss\fcharset177\fprq2 Arial (Hebrew);}{\f277\fbidi \fswiss\fcharset178\fprq2 Arial
(Arabic);}{\f278\fswiss\fcharset186\fprq2 Arial Baltic;}{\f279\fswiss\fcharset163\fprq2 Arial
(Vietnamese);}
{\f281\fmodern\fcharset238\fprq1 Courier New CE;}{\f282\fmodern\fcharset204\fprq1 Courier New
Cyr;}{\f284\fmodern\fcharset161\fprq1 Courier New Greek;}{\f285\fmodern\fcharset162\fprq1 Courier
New Tur;}
{\f286\fbidi \fmodern\fcharset177\fprq1 Courier New (Hebrew);}{\f287\fbidi \fmodern\fcharset178\fprq1
Courier New (Arabic);}{\f288\fmodern\fcharset186\fprq1 Courier New Baltic;}{\f289\ ... {content
removed}
```

Low **Possible Server Path Disclosure (win32)**

File Names:

- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/comment.aspx
- http://demo.testfire.net:80/bank/login.aspx
- http://demo.testfire.net:80/bank/queryxpath.aspx
- http://demo.testfire.net:80/bank/customize.aspx
- http://demo.testfire.net:80/feedback.aspx
- http://demo.testfire.net:80/servererror.aspx?aspxerrorpath=/Web.Config
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx
- http://demo.testfire.net:80/subscribe.aspx

Summary:

A minor vulnerability has been detected within your web application due to the discovery of a fully qualified path name to the root of your system. This most often occurs in context of an error being produced by the web application. Fully qualified server path names allow an attacker to know the file system structure of the web server, which is a baseline for many other types of attacks to be successful. Recommendations include

adopting a consistent error handling scheme and mechanism that prevents fully qualified path names from being displayed.

Fix:

For Development:

Don't display fully qualified pathnames as part of error or informational messages. At the least, fully qualified pathnames can provide an attacker with important information about the architecture of web application.

For Security Operations:

The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? It is often a seemingly innocuous piece of information that provides an attacker with the means to discover something else which he can then utilize when conducting an attack.

Attack Request:

```
POST /bank/login.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 90
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641

uid=12345'+and++(select+count(*)+from+spitable)%3d1+or+'1'%3d'0+&passw=foo&btnSubmit=Login
```

Attack Response:

```
HTTP/1.1 500 Internal Server Error
Connection: close
Date: Wed, 04 Feb 2009 06:44:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 5181



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0_head"><title>
 Altoro Mutual: Server Error
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link
href="../style.css" rel="stylesheet" type="text/css" /></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0_HyperLink1" href="../default.aspx"
style="height:80px;width:183px;"><img src="../images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
```

```
id="_ctl0_HyperLink3" href="../default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0_HyperLink4" href="../feedback.aspx">Feedback</a> | <label for="txtSearch">Search</label>
        <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
        <input type="submit" value="Go" />
    </td>
   </tr>
   <tr>
    <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0_Image1" src="../images/header_pic.jpg" border="0" style="height:60px;width:354px;" /></td>
   </tr>
  </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">


<div class="err" style="width: 99%;">

<h1>An Error Has Occurred</h1>

< ... {content removed}
```

Info **Default Page (default.aspx)**

File Names:
- http://demo.testfire.net:80/Default.aspx

Summary: The predictably named file /Default.aspx was found. Recommendations include evaluating whether this file should be publicly accessible.

Implication: The implications of this file depend on what it contains. Often, it is a seemingly innocuous piece of information that can complete the knowledge a potential attacker needs to compromise a site.

Fix:

For Security Operations:
Evaluate whether the presence of the discovered file is intended and should be directly accessible to remote users. Restrict access to important directories or files.

For Development:
Make sure the naming conventions of your file system do not reveal information of value to a potential attacker.

For QA:
Check to make sure things of value to a potential attacker have not been left publicly available.

Reference:

Apache:
Security Tips for Server Configuration
Protecting Confidential Documents at Your Site
Securing Apache - Access Control

IIS:
Implementing NTFS Standard Permissions on Your Web Site

Netscape:
Controlling Access to Your Server

General:
Password-protecting web pages
Web Security

Attack Request:

```
GET /Default.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:41:40 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 9605



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range
of commercial, private, retail and mortgage banking services to small and middle-market businesses and
individuals."></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="default.aspx"
style="height:80px;width:183px;"><img src="images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="bank/login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
         <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
         <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
 ... {content removed}
```

Info **OPTIONS Method Supported**

File Names:
- http://demo.testfire.net:80/

Summary: The server supports the OPTIONS HTTP method. The OPTIONS method is used to determine what other methods the server supports for a given URI/resource.

Reference: **RFC 2616 Section 9: HTTP Methods:**
http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Apache:
Apache HTTP Server Version 2.0
Apache HTTP Server Version 1.3

Microsoft:
UrlScan Security Tool
How to configure the URLScan Tool
Setting Application Mappings in IIS 6.0

Attack Request:
```
OPTIONS / HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ou
1nnj33b5w02x55ctl3kbed;amSessionId=037914641
```

Attack Response:
```
HTTP/1.1 200 OK
Allow: OPTIONS, TRACE, GET, HEAD
Content-Length: 0
Server: Microsoft-IIS/6.0
Public: OPTIONS, TRACE, GET, HEAD, POST
X-Powered-By: ASP.NET
Date: Wed, 04 Feb 2009 06:40:56 GMT
```

Info **Flash Object Detected**

File Names:
- http://demo.testfire.net:80/default.aspx?content=inside_contact.htm

Summary: A Flash movie or Flash object was found. Flash movies and objects can be decompiled and may contain sensitive information. An attacker could decompile the Flash file and gain access to the confidential information, including any hard-coded passwords and keys, within the Flash file.

Execution: A primary tool in the arsenal of the attacker who wants to get inside your code is the decompiler. A decompiler takes an executable file and attempts to re-create the original source code. It may be almost impossible to go from machine code to a high-level language. It is, however, easy to recover an assembly language version of the program.

Implication: The attacker's goal in re-creating the original source code may include one or more of the following:
- To steal a valuable algorithm for use in his own code
- To understand how a security function works to enable him to bypass it
- To extract confidential information, such as hard-coded passwords and keys

- To enable him to alter the code so that it behaves in a malicious way

Reference: Flare - Flash Decompiler
http://www.nowrap.de/flare.html

Attack Request:
```
GET /default.aspx?content=inside_contact.htm HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632
```

Attack Response:
```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:37:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 10442



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range
of commercial, private, retail and mortgage banking services to small and middle-market businesses and
individuals."></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="default.aspx"
style="height:80px;width:183px;"><img src="images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="bank/login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
         <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
         <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
 ... {content removed}
```

Info **WSDL File Discovered**

File Names:
- http://demo.testfire.net:80/bank/ws.asmx?WSDL

Summary: A web services description document has been discovered on the server. An attacker can use this information to gain unauthorized access to critical server data.

Execution: Click http://demo.testfire.net:80/bank/ws.asmx?WSDL to verify the vulnerability in a web browser.

Implication: WSDL's contain information about the web services that the server offers. An unintended exposure of this information can cause unauthorized access to server methods. An attacker can bypass the client application and directly call web methods.

Fix: It is recommended that you perform a security assessment of the discovered web services to determine if any vulnerabilities are present.

Attack Request:
```
GET /bank/ws.asmx?WSDL HTTP/1.1
Referer: http://demo.testfire.net:80/bank/ws.asmx
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
```

Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecxsef55bmimsp55ocuoszme;amSessionId=037614632;lang=

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:38:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: private, max-age=0
Content-Type: text/xml; charset=utf-8
Content-Length: 8329

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
xmlns:tns="http://www.altoromutual.com/bank/ws/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
targetNamespace="http://www.altoromutual.com/bank/ws/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">Core services offered by Altoro
Mutual bank.</wsdl:documentation>
  <wsdl:types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://www.altoromutual.com/bank/ws/">
      <s:element name="IsValidUser">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="UserId" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="IsValidUserResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="IsValidUserResult" type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetUserAccounts">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="UserId" type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetUserAccountsResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetUserAccountsResult"
type="tns:ArrayOfAccountData" />
          </s:sequence>
  ... {content removed}
```

Best Practice **Hidden Form Value**

File Names:
- http://demo.testfire.net:80/feedback.aspx

Summary: While preventing display of information on the web page itself, the information submitted via hidden form fields is easily accessible, and could give an attacker valuable information that would prove helpful in escalating his attack methodology. Recommendations include not relying on hidden form fields as a security solution for any area of the web application that contains sensitive information or access to privileged functionality such as remote site administration functionality.

Execution: Any attacker could bypass a hidden form field security solution by viewing the source code of that particular page.

Implication: The greatest danger from exploitation of a hidden form field design vulnerability is that the attacker will gain information that will help in orchestrating a far more dangerous attack.

Fix: Do not rely on hidden form fields as a method of passing sensitive information or maintaining session state. One workable bypass is to encrypt the hidden values in a form, and then decrypt them when that information is to be utilized by a database operation or a script. From a security standpoint, the best method of temporarily storing information required by different forms is to utilize a session cookie.

Whether hidden or not, if your site utilizes values submitted via a form to construct database queries, do not make the assumption that the data is non-malicious. Instead, utilize the following recommendations to sanitize user supplied input.

- Stringently define the data type (for instance, a string, an alphanumeric character, etc) that the application will accept.
- Use what is good instead of what is bad.
- Validate input for improper characters.
- Do not display error messages to the end user that provide information (such as table names) that could be utilized in orchestrating an attack.
- Define the allowed set of characters. For instance, if a field is to receive a number, only let that field accept numbers.
- Define the maximum and minimum data lengths for what the application will accept.
- Specify acceptable numeric ranges for input.

Attack Request:

```
GET /feedback.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:37:11 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 8721



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual: Feedback
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
rel="stylesheet" type="text/css" /><meta name="keywords" content="Altoro Muutual, feedback, contact
us"></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="default.aspx"
style="height:80px;width:183px;"><img src="images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
       <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="bank/login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
         <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
         <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
   </table>
 </form>
</div>

<div id="wrapper" style="width: 99%;">


<table cellspacing="0" wid ... {content removed}
```

Best Practice **Set-Cookie does not use HTTPOnly Keyword**

File Names:

- http://demo.testfire.net:80/bank/account.aspx
- http://demo.testfire.net:80/bank/apply.aspx

- http://demo.testfire.net:80/bank/customize.aspx
- http://demo.testfire.net:80/bank/main.aspx
- http://demo.testfire.net:80/bank/transaction.aspx
- http://demo.testfire.net:80/bank/transfer.aspx

Summary: The web application does not utilize HTTP only cookies. This is a new security feature introduced by Microsoft in IE 6 SP1 to mitigate the possibility of a successful Cross-Site scripting attack by not allowing cookies with the HTTP only attribute to be accessed via client-side scripts. Recommendations include adopting a development policy that includes the utilization of HTTP only cookies, and performing other actions such as ensuring proper filtration of user-supplied data, utilizing client-side validation of user supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Reference: **References:**
http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/httponly_cookies.asp

Attack Request:
```
GET /bank/account.aspx HTTP/1.1
Referer: http://demo.testfire.net:80/bank/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632
```

Attack Response:
```
HTTP/1.1 302 Found
Date: Wed, 04 Feb 2009 06:38:01 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Location: /bank/login.aspx
Set-Cookie: amUserId=; expires=Tue, 03-Feb-2009 06:38:01 GMT; path=/
Set-Cookie: amCreditOffer=; expires=Tue, 03-Feb-2009 06:38:01 GMT; path=/
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 133

<html><head><title>Object moved</title></head><body>
<h2>Object moved to <a href="/bank/login.aspx">here</a>.</h2>
</body></html>
```

Best Practice **Vulnerable Flash Engine Allowed**

File Names:
- http://demo.testfire.net:80/default.aspx?content=inside_contact.htm

Summary: A Flash movie or Flash object was found and did not require the latest version of the Flash Player. The Flash Player itself has had many vulnerabilities reported against it. Allowing your website users to use an old and unsupported Flash Player introduces unnecessary risk. Generally web developers will allow the minimum required version run the flash object, but this exposes the website user to an exploitable situation. Requiring the latest Flash Player version will ensure your users are protected from known flaws in the player itself; these past vulnerabilities include Cross-Site Scripting, and Remote Code Execution (on the client machine). Recommendations include requiring the latest version of the Flash Player be utilized.

Implication: The Flash Player has had many vulnerabilities reported against it since its inception. These vulnerabilities can be leveraged to attack web users through such vectors as Cross-Site Scripting which in turn can be used to execute binary applications on the clients machine. Since the Flash Player is backward compatible requiring a higher version than the Flash Object requires will not affect the exeution or processing of the Flash Object.

Fix: The Flash Object declaration in html allows a webdeveloper to specify the version of flash required to play the Flash Object embedded in the page.

Adding the attribute:

codebase="http://fpdownload.adobe.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,124,0"

to the object tag in html will communicate to the users flash player that version 9.0.124.0 is required to run this flash object. If the user's Flash Player is prior to that version, the user will be warned and a dialog will prompt him to visit Adobe's Flash Player page and update to the latest version.

To find the latest version of Flash Player visit this page
http://www.adobe.com/shockwave/download/download.cgi?P1_Prod_Version=shockwaveflash

Attack Request:
```
GET /default.aspx?content=inside_contact.htm HTTP/1.1
Referer: http://demo.testfire.net:80/
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)
Accept: */*
Pragma: no-cache
Host: demo.testfire.net
Connection: Keep-Alive
Cookie:
CustomCookie=WebInspect32791ZXF6B5BF2AB6E547259AEC366BE89E3589YB6BB;ASP.NET_SessionId=ecx
sef55bmimsp55ocuoszme;amSessionId=037614632
```

Attack Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Feb 2009 06:37:10 GMT
Server: Microsoft-IIS/6.0
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 10442



<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
<head id="_ctl0__ctl0_head"><title>
 Altoro Mutual
</title><meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"><link href="style.css"
rel="stylesheet" type="text/css" /><meta name="description" content="Altoro Mutual offers a broad range
of commercial, private, retail and mortgage banking services to small and middle-market businesses and
individuals."></head>
<body style="margin-top:5px;">

<div id="header" style="margin-bottom:5px; width: 99%;">
  <form id="frmSearch" method="get" action="/search.aspx">
   <table width="100%" border="0" cellpadding="0" cellspacing="0">
    <tr>
       <td rowspan="2"><a id="_ctl0__ctl0_HyperLink1" href="default.aspx"
style="height:80px;width:183px;"><img src="images/logo.gif" border="0" /></a></td>
     <td align="right" valign="top">
      <a id="_ctl0__ctl0_LoginLink" title="It does not appear that you have properly authenticated yourself.
Please click here to sign in." href="bank/login.aspx" style="color:Red;font-weight:bold;">Sign In</a> | <a
id="_ctl0__ctl0_HyperLink3" href="default.aspx?content=inside_contact.htm">Contact Us</a> | <a
id="_ctl0__ctl0_HyperLink4" href="feedback.aspx">Feedback</a> | <label
for="txtSearch">Search</label>
        <input type="text" name="txtSearch" id="txtSearch" accesskey="S" />
        <input type="submit" value="Go" />
     </td>
    </tr>
    <tr>
     <td align="right" style="background-image:url(/images/gradient.jpg);padding:0px;margin:0px;"><img
id="_ctl0__ctl0_Image1" src="images/header_pic.jpg" border="0" style="height:60px;width:354px;"
/></td>
    </tr>
 ... {content removed}
```